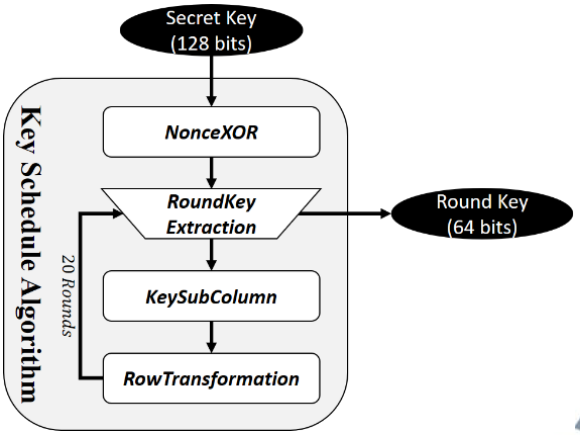
APPENDIX A

Key Schedule Algorithm Components




```
NonceXOR (SecretKey)
RoundKeyExtraction (KeyState)
for i = 1 to 20
{
    KeySubColumn (KeyState);
    RowTransformation (KeyState);
    RoundKeyExtraction (KeyState);
}
```

Figure 1: Key Schedule Algorithm

Input A = ***SecretKey*** 128-bit (32-hex):
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 *binary*
(*00000000000000000000000000000000*) *hexadecimal*

Input B = ***Nonce*** 128-bit (32-hex):
01000001010000100100010001010101010011000100000101001100010010010100011001011010010000010100100101101000001010100100100100101000001 *binary*
(*414244554C414C49465A414B41524941*) *hexadecimal*

1) **Process 1: Nonce XOR**

Input(1_a) = ***SecretKey***:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 (*00000000000000000000000000000000*)

Input(1_b) = ***Nonce***:
01000001010000010010001000101010101001100010000001010011000100100101000110010110100100000101000101101000000101010010010010010100000001 (*414244554C414C49465A414B41524941*)

i) **Step 1**: **Input(1_a)** XOR **Input(1_b)**.

Output(1):
01000001010000010010001000101010101001100010000001010011000100100101000110010110100100000101000101101000000101010010010010010100000001
(*414244554C414C49465A414B41524941*)

2) **Process 2: Key State Conversion**

Input(2) = **Output(1):**
01000001010000010010001000101010101001100010000001010011000100100101000110010110100100000101000101101000000101010010010010010100000001 (*414244554C414C49465A414B41524941*)

i) **Step 1:** Divide **Input(2)** into 4 blocks.

Output(2_a) = 1st 32 bits of **Input(2)**: 01000001010000010010001000101010 (*41424455*)
Output(2_b) = 2nd 32 bits of **Input(2)**: 01001100010000001010011000100100 1 (*4C414C49*)
Output(2_c) = 3rd 32 bits of **Input(2)**: 01000110010110100100000101001011 (*465A414B*)
Output(2_d) = 4th 32 bits of **Input(2)**: 01000001010100100100100101000001 (*41524941*)

ii) **Step 2:** Sort **Output(2_a)**, **Output(2_b)**, **Output(2_c)**, and **Output(2_d)** in reverse order to obtain the ***RowKeys***.

Output(2_A) = Sort **Output(2_a)** = 10101010001000100100001010000010 (*AA224282*): ***RowKey*$_3$**
Output(2_B) = Sort **Output(2_b)** = 10010010001100101000001000110010 (*92328232*): ***RowKey*$_2$**
Output(2_C) = Sort **Output(2_c)** = 11010010100000100101101001100010 (*D2825A62*): ***RowKey*$_1$**
Output(2_D) = Sort **Output(2_d)** = 10000010100100100100101010000010 (*82924A82*): ***RowKey*$_0$**

3) **Process 3: Initial Round Key Extraction (*RoundKey*$_0$)**

Input(3_a) = ***RowKey*$_3$**: 10101010001000100100001010000010 (*AA224282*)
Input(3_b) = ***RowKey*$_2$**: 10010010001100101000001000110010 (*92328232*)
Input(3_c) = ***RowKey*$_1$**: 11010010100000100101101001100010 (*D2825A62*)
Input(3_d) = ***RowKey*$_0$**: 10000010100100100100101010000010 (*82924A82*)

i) **Step 1:** Take 16 rightmost bits of **Input(3_a)**, **Input(3_b)**, **Input(3_c)**, and **Input(3_d)**.

Output(3_a) = 16 rightmost bits of **Input(3_a):** 0100001010000010 (*4282*)
Output(3_b) = 16 rightmost bits of **Input(3_b):** 1000001000110010 (*8232*)
Output(3_c) = 16 rightmost bits of **Input(3_c):**0101101001100010 (*5A62*)
Output(3_d) = 16 rightmost bits of **Input(3_d):** 0100101010000010 (*4A82*)

ii) **Step 2:** Append **Output(3_a)**, **Output(3_b)**, **Output(3_c)**, and **Output(3_d)**.

Output(3) = 01000010100000101000001000110010010110100110001001001010100000010
(*42828232*5A624A82*): ***RoundKey*$_0$**

4) **Process 4: Key Sub Column**

Input(4_a) = ***RowKey*$_3$**: 10101010001000100100001010000010 (*AA224282*)
Input(4_b) = ***RowKey*$_2$**: 10010010001100101000001000110010 (*92328232*)
Input(4_c) = ***RowKey*$_1$**: 11010010100000100101101001100010 (*D2825A62*)
Input(4_d) = ***RowKey*$_0$**: 10000010100100100100101010000010 (*82924A82*)
* For the following rounds, **Input(4_a)**, **Input(4_b)**, **Input(4_c)**, and **Input(4_d)** are replaced with the generated ***RowKeys*** from previous round.

i) **Step 1**: Take 8 rightmost bits of **Input(4_a)**, **Input(4_b)**, **Input(4_c)**, and **Input(4_d)**.

Output(4_a) = 8 rightmost bits of **Input(4_a):** 10000010 (*82*)
Output(4_b) = 8 rightmost bits of **Input(4_b):** 00110010 (*32*)
Output(4_c) = 8 rightmost bits of **Input(4_c):** 01100010 (*62*)
Output(4_d) = 8 rightmost bits of **Input(4_d):** 10000010 (*82*)

ii) **Step 2**: Append 1 bit each from **Output(4_a)**, **Output(4_b)**, **Output(4_c)**, and **Output(4_d)** to obtain 4 bits output. Then, convert the output to a decimal value. Repeat for 8 times.

Output(4_e) = 1001: 9
Output(4_f) = 0010: 2
Output(4_g) = 0110: 6
Output(4_h) = 0100: 4
Output(4_i) = 0000: 0
Output(4_j) = 0000: 0
Output(4_k) = 1111: 15
Output(4_l) = 0000: 0

iii) **Step 3**: Replace **Output(4_e)**, **Output(4_f)**, **Output(4_g)**, **Output(4_h)**, **Output(4_i)**, **Output(4_j)**, **Output(4_k)**, and **Output(4_l)** with PRESENT S-box value as shown in Table 1. Then, convert the output to binary.

Table 1: PRESENT S-box

X (Input)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$ (Output)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Output(4_m) = Output(4_e) → S-box = 14: 1110
Output(4_n) = Output(4_f) → S-box = 6: 0110
Output(4_o) = Output(4_g) → S-box = 10: 1010
Output(4_p) = Output(4_h) → S-box = 9: 1001
Output(4_q) = Output(4_i) → S-box = 12: 1100
Output(4_r) = Output(4_j) → S-box = 12: 1100
Output(4_s) = Output(4_k) → S-box = 2: 0010
Output(4_t) = Output(4_l) → S-box = 12: 1100

iv) **Step 4**: Append 1 bit each from **Output(4_m)**, **Output(4_n)**, **Output(4_o)**, **Output(4_p)**, **Output(4_q)**, **Output(4_r)**, **Output(4_s)**, and **Output(4_t)** to obtain 8 bits output. Repeat for 4 times.

Output(4_u): 10111101 (*BD*)
Output(4_v): 11001101 (*CD*)
Output(4_w): 11100010 (*E2*)
Output(4_x): 00010000 (*10*)

v) **Step 5**: Replace **Output(4_u)**, **Output(4_v)**, **Output(4_w)**, **Output(4_x)** at the 8 rightmost bits of ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$** accordingly. Update the new ***RowKeys***.

Output(4_A) = 10101010001000100100001010111101 (*AA2242BD*): ***RowKey*$_3$**
Output(4_B) = 10010010001100101000001011001101 (*923282CD*): ***RowKey*$_2$**
Output(4_C) = 11010010100000100101101011100010 (*D2825AE2*): ***RowKey*$_1$**
Output(4_D) = 10000010100100100100101000010000 (*82924A10*): ***RowKey*$_0$**

5) **Process 5: Row Transformation**

Input(5_a) = ***RowKey*$_3$**: 10101010001000100100001010111101 (*AA2242BD*)
Input(5_b) = ***RowKey*$_2$**: 10010010001100101000001011001101 (*923282CD*)
Input(5_c) = ***RowKey*$_1$**: 11010010100000100101101011100010 (*D2825AE2*)
Input(5_d) = ***RowKey*$_0$**: 10000010100100100100101000010000 (*82924A10*)

i) **Step 1**: $\boldsymbol{RowKey'_3 = RowKey_0}$.

Output(5_a) = Replace ***RowKey*$_3$** with ***RowKey*$_0$**
= 10000010100100100100101000010000 (*82924A10*): ***RowKey*$_3$**

ii) **Step 2:** $RowKey'_2 = (RowKey_2 <<< 16) \oplus RowKey_3$.

Output(5_b) = Rotate ***RowKey*$_2$** 16 bits to the left:
10000010110011011001001000110010 (*82CD9232*)

Output(5_c) = **Output(5_b)** XOR ***RowKey*$_3$**
= 00101000111011111101000010001111 (*28EFD08F*): ***RowKey*$_2$**

iii) **Step 3:** $RowKey'_1 = RowKey_2$.

Output(5_d) = ***RowKey*$_2$** = 10010010000110010100000101100110 1 (*923282CD*): ***RowKey*$_1$**

iv) **Step 4:** $RowKey'_0 = (RowKey_0 <<< 8) \oplus RowKey_1$.

Output(5_e) = Rotate ***RowKey*$_0$** 8 bits to the left: 10010010010010100001000010000010
(*924A1082*)

Output(5_f) = **Output(5_e)** XOR ***RowKey*$_1$**
= 01000000110010000100101001100000 (*40C84A60*): ***RowKey*$_0$**

v) **Step 5:** Update the new ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$**.

Output(5_A) = ***RowKey*$_3$**: 10000010100100100100101000010000 (*82924A10*)
Output(5_B) = ***RowKey*$_2$**: 00101000111011111101000010001111 (*28EFD08F*)
Output(5_C) = ***RowKey*$_1$**: 10010010000110010100000101100110 1 (*923282CD*)
Output(5_D) = ***RowKey*$_0$**:01000000110010000100101001100000 (*40C84A60*)

6) **Process 6: Round Key Extraction (*RoundKey*$_1$ to *RoundKey*$_{20}$)**

Input(6_a) = ***RowKey*$_3$**: 10000010100100100100101000010000 (*82924A10*)
Input(6_b) = ***RowKey*$_2$**: 00101000111011111101000010001111 (*28EFD08F*)
Input(6_c) = ***RowKey*$_1$**: 10010010000110010100000101100110 1 (*923282CD*)
Input(6_d) = ***RowKey*$_0$**: 01000000110010000100101001100000 (*40C84A60*)

i) **Step 1**: Take 16 rightmost bits of **Input(6_a)**, **Input(6_b)**, **Input(6_c)**, and **Input(6_d)**.

Output(6_a) = 16 rightmost bits of **Input(6_a):** 0100101000010000 (*4A10*)
Output(6_b) = 16 rightmost bits of **Input(6_b):** 1101000010001111 (*D08F*)
Output(6_c) = 16 rightmost bits of **Input(6_c):**1000001011001101 (*82CD*)
Output(6_d) = 16 rightmost bits of **Input(6_d):** 0100101001100000 (*4A60*)

ii) **Step 2:** Append **Output(6_a)**, **Output(6_b)**, **Output(6_c)**, and **Output(6_d)**.

Output(6) = 0100101000010000110100001000111110000010110011010100101001100000
(*4A10D08F82CD4A60*): ***RoundKey*$_1$**

* **Output(6)** in this example is the generated ***RoundKey*$_1$**.

7) **Process 7**: Generate the remaining ***RoundKeys*** (***RoundKey*$_2$** to ***RoundKey*$_{20}$**)

i) **Step 1:** Repeat **Process 4**, **Process 5**, and **Process 6** for another 19 times to generate ***RoundKey*$_2$** to ***RoundKey*$_{20}$**.

a) **Process 4** (Key Sub Column)
Input: ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$** from **Process 5** (previous round Row Transformation).
Output: Updated ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$**.

b) **Process 5** (Row Transformation)
Input: ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$** from **Process 4** (current round Key Sub Column).
Output: Updated ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$**.

c) **Process 6** (RoundKey Extraction)
Input: ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$** from **Process 5** (current round Row Transformation).
Output: ***RoundKeys***.

ii) **Step 2**: Store all ***RoundKeys*** (***RoundKey*$_0$** to ***RoundKey*$_{20}$**).

Output(7$_A$) = 0100001010000010100000100011001001011010011000100100101010000010
(*428282325A624A82*): ***RoundKey*$_0$**

Output(7$_B$) = 0100101000010000110100001000111110000010110011010100101001100000
(*4A10D08F82CD4A60*): ***RoundKey*$_1$**

Output(7$_C$) = 0100101001110010011000100010000011010000001000001111000010011101
(*4A726220D020F09D*): ***RoundKey*$_2$**

Output(7$_D$) = 1111000011101111000110001010001001100010100111010011111100001000
(*F0EF18A2629D3F08*): ***RoundKey*$_3$**

Output(7$_E$) = 0011111111000101110100100101100000011000001101111010011100000101
(*3FC5D2581837A705*): ***RoundKey*$_4$**

Output(7$_F$) = 1010011110001000011111010101001011010010011000101001000011111111
(*A7887D52D26290FF*): ***RoundKey*$_5$**

Output(7$_G$) = 1001000001100111001001011001011101111101110011111011010100000111
(*906725977DCFB507*): ***RoundKey*$_6$**

Output(7_H) = 101101010111000010100000100100100010010101001000000011010101111
(*B570A09225480D5F*): ***RoundKey*$_7$**

Output(7_I) = 000011011011110100111111011100101010000000010101100110000110010 1
(*0DBD3F72A0159865*): ***RoundKey*$_8$**

Output(7_J) = 100110001011101000011101110011010011111100100000000110100101000 0
(*98BA1DCD3F201A50*): ***RoundKey*$_9$**

Output(7_K) = 000110100010011111111010110111010001110110111000000110001111011 1
(*1A27FADD1DB818F7*): ***RoundKey*$_{10}$**

Output(7_L) = 000110001001010111111111010011011111101010010111100010000001111 01
(*1895FF4DFA97883D*): ***RoundKey*$_{11}$**

Output(7_M) = 100010001110000000100111011001011111111111010001000011010001101 01
(*88E02765FFA21A35*): ***RoundKey*$_{12}$**

Output(7_N) = 000110101001000001000101111110100010011111011010011011111011011 1
(*1A9045FA27DA6FB7*: ***RoundKey*$_{13}$**

Output(7_O) = 011011110000011110001111011000000100010100000111001000000100110 00
(*6F078F6045072098*): ***RoundKey*$_{14}$**

Output(7_P) = 001000001111111100111111000111111000111110011111101110100110010 1
(*20FF3F1F8F9FBA65*): ***RoundKey*$_{15}$**

Output(7_Q) = 101110101001101011101111111010000001111111110000000001010101101 101
(*BA9AEFD03FE0156D*): ***RoundKey*$_{16}$**

Output(7_R) = 000101011110011111100111100101011110111101011011011011000011100 00
(*15E7E795EF5DD870*): ***RoundKey*$_{17}$**

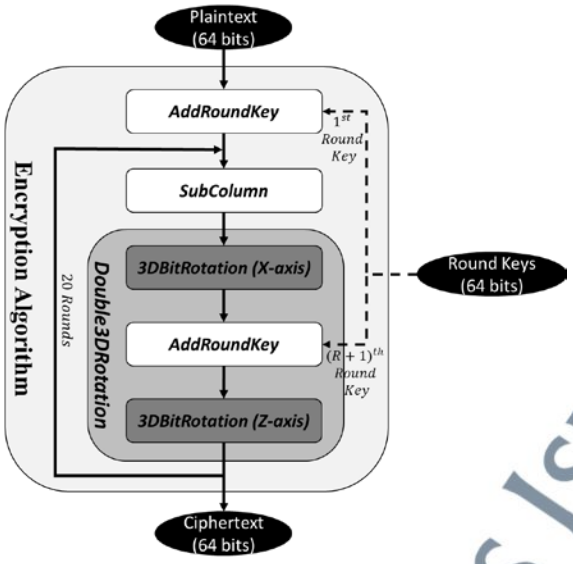
Output(7_S) = 110110000001011101110010001001111110011110111000011111000000100 111
(*D8177227E7B8F827*): ***RoundKey*$_{18}$**

Output(7_T) = 111110000011011101100101000010100111001011111111110100000111001 11
(*F837650A72FFD0E7*): ***RoundKey*$_{19}$**

Output(7_U) = 110100001101000010000111001001110110010100010000101000101001100 00
(*D0D087276510A298*): ***RoundKey*$_{20}$**

APPENDIX B

Encryption Algorithm Components



RoundKeysGeneration (*Key*)
AddRoundKey (*CipherState*, $RoundKey_0$)
for i = 1 *to* 20
{
    SubColumn (*CipherState*);
    Double3DRotation (*CipherState*, $RoundKey_i$);
}

Figure 1: Encryption Algorithm

Input A = ***Plaintext*** 64-bit (16-hex):
0000000000000000000000000000000000000000000000000000000000000000 *binary*
(*0000000000000000*) *hexadecimal*

Input B = ***SecretKey*** 128-bit (32-hex):
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000 *binary*
(*00000000000000000000000000000000*) *hexadecimal*

Input C = ***Nonce*** 128-bit (32-hex):
01000001010000100100010001010101010011000100000101001100010010010100011001011010010000010100101101000001010100100100100101000001 *binary*
(*414244554C414C49465A414B41524941*) *hexadecimal*

1) **Process 1: Add Round Key**

Input(1_a) = ***Plaintext***:
00000000000000000000000000000000000000000000000000000000000000000000000000000000 (*0000000000000000*)

Input(1_b) = ***SubKey$_0$***:
01000001010000010100000100011001001011010011000100100101010000010 (*428282325A624A82*)

i) **Step 1: Input(1_a) XOR Input(1_b).**

Output(1): 01000001010000010100000100011001001011010011000100100101010000010
(*428282325A624A82*)

2) **Process 2: Cipher State Conversion**

Input(2) = **Output(1):**
01000001010000010100000100011001001011010011000100100101010000010 (*428282325A624A82*)

i) **Step 1:** Divide **Input(2)** into 4 blocks.

Output(2_a) = 1st 16 bits of **Input(2)**: 0100001010000010 (*4282*) : ***RowKey$_3$***
Output(2_b) = 2nd 16 bits of **Input(2)**: 1000001000110010 (*8232*) : ***RowKey$_2$***
Output(2_c) = 3rd 16 bits of **Input(2)**: 0101101001100010 (*5A62*) : ***RowKey$_1$***
Output(2_d) = 4th 16 bits of **Input(2)**: 0100101010000010 (*4A82*) : ***RowKey$_0$***

3) **Process 3: Sub Column**

Input(3_a) = ***RowKey$_3$***: 0100001010000010 (*4282*)
Input(3_b) = ***RowKey$_2$***: 1000001000110010 (*8232*)
Input(3_c) = ***RowKey$_1$***: 0101101001100010 (*5A62*)
Input(3_d) = ***RowKey$_0$***: 0100101010000010 (*4A82*)

* For the following rounds, **Input(3_a)**, **Input(3_b)**, **Input(3_c)**, and **Input(3_d)** are replaced with the generated ***Ciphertext*** in the form of ***RowKeys*** from previous round.

i) **Step 1**: Append 1 bit each from **Input(3_a)**, **Input(3_b)**, **Input(3_c)**, and **Input(3_d)** to obtain 4 bits output. Then, convert the output to a decimal value. Repeat for 16 times.

Output(3_{e1}) = 0100: 4
Output(3_{e2}) = 1011: 11
Output(3_{e3}) = 0000: 0
Output(3_{e4}) = 0010: 2
Output(3_{e5}) = 0011: 3
Output(3_{e6}) = 0000: 0
Output(3_{e7}) = 1111: 15
Output(3_{e8}) = 0000: 0
Output(3_{e9}) = 1001: 9

Output(3_{e10}) = 0010: 2
Output(3_{e11}) = 0110: 6
Output(3_{e12}) = 0100: 4
Output(3_{e13}) = 0000: 0
Output(3_{e14}) = 0000: 0
Output(3_{e15}) = 1111: 15
Output(3_{e16}) = 0000: 0

ii) **Step 2**: Replace **Output(3_{e1})** to **Output(3_{e16})** with the PRESENT S-box value as shown in Table 1. Then, convert the output to binary.

Table 1: PRESENT S-box

X (Input)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(*x*) (Output)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Output(3_{f1}) = **Output(3_{e1})** → S-box = 9: 1001
Output(3_{f2}) = **Output(3_{e2})** → S-box = 8: 1000
Output(3_{f3}) = **Output(3_{e3})** → S-box = 12: 1100
Output(3_{f4}) = **Output(3_{e4})** → S-box = 6: 0110
Output(3_{f5}) = **Output(3_{e5})** → S-box = 11: 1011
Output(3_{f6}) = **Output(3_{e6})** → S-box = 12: 1100
Output(3_{f7}) = **Output(3_{e7})** → S-box = 2: 0010
Output(3_{f8}) = **Output(3_{e8})** → S-box = 12: 1100
Output(3_{f9}) = **Output(3_{e9})** → S-box = 14: 1110
Output(3_{f10}) = **Output(3_{e10})** → S-box = 6: 0110
Output(3_{f11}) = **Output(3_{e11})** → S-box = 10: 1010
Output(3_{f12}) = **Output(3_{e12})** → S-box = 9: 1001
Output(3_{f13}) = **Output(3_{e13})** → S-box = 12: 1100
Output(3_{f14}) = **Output(3_{e14})** → S-box = 12: 1100
Output(3_{f15}) = **Output(3_{e15})** → S-box = 2: 0010
Output(3_{f16}) = **Output(3_{e16})** → S-box = 12: 1100

iii) **Step 3**: Append 1 bit each from **Output(3_{f1})** to **Output(3_{f16})** to obtain 16 bits output. Repeat for 4 times. Update the new ***RowKeys***.

Output(3_g): 1110110110111101 (*EDBD*): ***RowKey*$_3$**
Output(3_h): 0011010111001101 (*35CD*): ***RowKey*$_2$**
Output(3_i): 0001101011100010 (*1AE2*): ***RowKey*$_1$**
Output(3_j): 1000100000010000 (*8810*): ***RowKey*$_0$**

4) **Process 4: 3D Bit Rotation (*X*-axis)**

Input(4_a) = ***RowKey*$_3$**: 1110110110111101 (*EDBD*)
Input(4_b) = ***RowKey*$_2$**: 0011010111001101 (*35CD*)
Input(4_c) = ***RowKey*$_1$**: 0001101011100010 (*1AE2*)
Input(4_d) = ***RowKey*$_0$**: 1000100000010000 (*8810*)

i) **Step 1**: Append **Input(4_a)**, **Input(4_b)**, **Input(4_c)**, and **Input(4_d)** to obtain 64 bits output.

Output(4_a) = 1110110110111101001101011100110100011010111000101000100000010000
(*EDBD35CD1AE28810*)

ii) **Step 2:** Permute the bits of **Output(4_a)** according to the position from the 3D Bit Rotation Permutation Table (*X*-axis) values as shown in Table 2.

Table 2: 3D Bit Rotation Permutation Table (X-axis)

Permutation Table (*X*-axis Rotation)							
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
28	24	20	16	29	25	21	17
30	26	22	18	31	27	23	19
47	46	45	44	43	42	41	40
39	38	37	36	35	34	33	32
51	55	59	63	50	54	58	62
49	53	57	61	48	52	56	60

Output(4_b) = **Output(4_a)** → P-table:
1110110110111101110011100001101101000111010110000010000000001100
(*EDBDCE1B4758200C*)

5) **Process 5: Add Round Key (3D Bit Rotation)**

Input(5_a) = **Output(4_b)**:
1110110110111101110011100001101101000111010110000010000000001100 (*EDBDCE1B4758200C*)

Input(5_b) = ***SubKey*$_1$**:
0100101000010000110100001000111110000010110011010100101001100000 (*4A10D08F82CD4A60*)

i) **Step 1: Input(5_a)** XOR **Input(5_b).**

Output(5):
1010011110101101000111101001010011000101100101010110101001101100
(*A7AD1E94C5956A6C*)

6) Process 6: 3D Bit Rotation (*Z*-axis)

Input(6) = **Output(5)**:
1010011110101101000111101001010011000101100101010110101001101100 (*A7AD1E94C5956A6C*)

i) Step 1: Permute the bits of **Input(6)** according to the position from the 3D Bit Rotation Permutation Table (*Z*-axis) values as shown in Table 3 to obtain the ***Ciphertext*** for current round.

Table 3: 3D Bit Rotation Permutation Table (Z-axis)

Permutation Table (*Z*-axis Rotation)							
0	13	62	51	4	29	58	35
8	45	54	19	12	61	50	3
16	9	46	55	20	25	42	39
24	41	38	23	28	57	34	7
32	5	30	59	36	21	26	43
40	37	22	27	44	53	18	11
48	1	14	63	52	17	10	47
56	33	6	31	60	49	2	15

Output(6) = **Input(6)** → P-table =
1100011011111110000010011000010111000101111100000000101101101111
(*C6FE0985C5F00B6F*): $\boldsymbol{Ciphertext_1}$
* **Output(6)** in this example is the generated $\boldsymbol{Ciphertext_1}$.

7) Process 7: Generate the remaining ***Ciphertext*** (**$Ciphertext_2$** to **$Ciphertext_{20}$**)

i) Step 1: Repeat **Process 3**, **Process 4**, **Process 5**, and **Process 6** for another 19 times to generate $\boldsymbol{Ciphertext_2}$ to $\boldsymbol{Ciphertext_{20}}$.

a) Process 3 (Sub Column)
Input: $\boldsymbol{Ciphertext_1}$ from **Process 6** (previous round 3D Bit Rotation *Z*-axis).
Output: Updated $\boldsymbol{RowKey_3}$, $\boldsymbol{RowKey_2}$, $\boldsymbol{RowKey_1}$, and $\boldsymbol{RowKey_0}$.

b) Process 4 (3D Bit Rotation *X*-axis)
Input: $\boldsymbol{RowKey_3}$, $\boldsymbol{RowKey_2}$, $\boldsymbol{RowKey_1}$, and $\boldsymbol{RowKey_0}$ from **Process 3** (current round Sub Column).
Output: $\boldsymbol{Output(4_b)}$.

c) Process 5 (Add Round Key)
Input: ***SubKey*** and $\boldsymbol{Output(4_b)}$ from **Process 4** (current round 3D Bit Rotation *X*-axis).
Output: **Output(5)**.

d) Process 6 (3D Bit Rotation *Z*-axis)
Input: **Output(5)** from **Process 5** (current round Add Round Key).
Output: ***Ciphertext***.

ii) **Step 2**: Obtain all rounds ***Ciphertext*** (***Ciphertext*$_1$** to ***Ciphertext*$_{20}$**).

Output(7_A) = 1100011011111111000001001100001011100010111111000000001011011011111
(*C6FE0985C5F00B6F*): ***Ciphertext*$_1$**

Output(7_B) = 1011110000011101101010001101011111101000000100101001010100010010
(*BC1DA8D7E8129512*): ***Ciphertext*$_2$**

Output(7_C) = 1111001101111111000100110001110100100100110100100010100000011111111
(*F37E263A49A4507F*): ***Ciphertext*$_3$**

Output(7_D) = 0001110011110100001001111010010000100110111011101100000000111111111
(*1CF427A426EEC07F*): ***Ciphertext*$_4$**

Output(7_E) = 1000011011000000001111110001010101111110101011011001001000000000100
(*86C07E2AFD5B2404*): ***Ciphertext*$_5$**

Output(7_F) = 0101101001111110000110101111101010010011010001101101100000011011
(*5A7E1AFA9346D81B*): ***Ciphertext*$_6$**

Output(7_G) = 0101011110000011011111011101101000010011110000101001011010000000
(*57837DDD09E14B40*): ***Ciphertext*$_7$**

Output(7_H) = 1010000011000010011110000100011111100011101000010000011111110011
(*A0C27847F1D083F3*): ***Ciphertext*$_8$**

Output(7_I) = 0101010100010110101110110001111101000110111011000110001001010100
(*5516BB1F46EC6254*): ***Ciphertext*$_9$**

Output(7_J) = 1110000000011011110010000010101001110011100111100111100111101010001000
(*E01BC82A739E7A88*): ***Ciphertext*$_{10}$**

Output(7_K) = 0111101001100110111010101011110000111000100001110000111110111101011
(*7A66D578710E1F7B*): ***Ciphertext*$_{11}$**

Output(7_L) = 0100010101001100111010110101100110101001101110000010011011111110
(*454CEB59A9B826FE*): ***Ciphertext*$_{12}$**

Output(7_M) = 1101010100000100001011100100100001110010101011000001011110110011
(*D5042E4872AC17B3*): ***Ciphertext*$_{13}$**

Output(7_N) = 0011011110101101110101101000111001100100100001000010001001000001
(*37ADD68E64842241*): ***Ciphertext*$_{14}$**

Output(7_O) = 1111100011011110110110010111000000111011001111001101101100001100
(*F8DED9703B3CDB0C*): ***Ciphertext*$_{15}$**

Output(7_P) = 1111101111010111101011101010111101101001100010110011111111000101
(*FBD7AEAF698B3FC5*): ***Ciphertext*$_{16}$**

Output(7_Q) = 0101011110111010101110001000010001000010011101101110110111000101
(*57BAB8844276EDC5*): ***Ciphertext*$_{17}$**

Output(7_R) = 1000100100110101010111001100101111011001000011101000101000100101
(*89355CCBD90E8A25*): ***Ciphertext*$_{18}$**

Output(7_S) = 0000100011001110010001100111001111001110010101111001010110000001
(*08CE4673CE579581*): ***Ciphertext*$_{19}$**

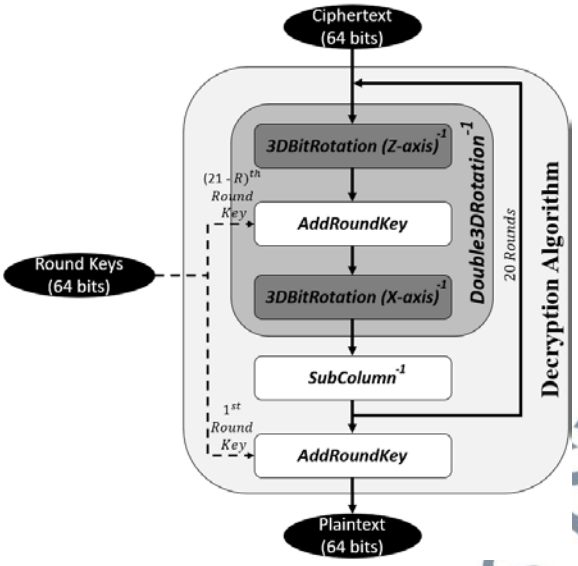
Output(7_T) = 0101111100000111111111000010111000100111001010010000101111110011 1
(*5F07F85C4E5217E7*): ***Ciphertext*$_{20}$**

iii) **<u>Step 3</u>:** Obtain ***Ciphertext*.**

Output(7) = **Output(7_T):**
0101111100000111111111000010111000100111001010010000101111110011 1
(*5F07F85C4E5217E7*): ***Ciphertext***

APPENDIX C

Decryption Algorithm Components



RoundKeysGeneration (*Key*)
for i = 1 *to* 20
{
Double3DRotation^{-1} (*CipherState*, *RoundKey*$_{21-i}$);
SubColumn^{-1} (*CipherState*);
}
AddRoundKey (*CipherState*, *RoundKey*$_0$)

Figure 1: Decryption Algorithm

Input A = ***Ciphertext*** 64-bit (16-hex):
0101111100000111111110000101110001001110010100100001011111100111 *binary*
(*5F07F85C4E5217E7*) *hexadecimal*

Input B = ***SecretKey*** 128-bit (32-hex):
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 *binary*
(*00000000000000000000000000000000*) *hexadecimal*

Input C = ***Nonce*** 128-bit (32-hex):
01000001010000100100010001010101010011000100000101001100010010010100011001011010010000010100101101000001010100100100100101000001 *binary*
(*414244554C414C49465A414B41524941*) *hexadecimal*

1) **Process 1: 3D Bit Rotation (*Z*-axis)**

Input(1) = ***Ciphertext*:**
0101111100000111111110000101110001001110010100100001011111100111 (*5F07F85C4E5217E7*)
* For the following rounds, **Input(1)** is replaced with the generated ***Plaintext*** from previous round.

i) **Step 1**: Permute the bits of **Input(1)** according to the position from the 3D Bit Rotation Inverse Permutation Table (*Z*-axis) values as shown in Table 1.

Table 1: 3D Bit Rotation Inverse Permutation Table (Z-axis)

Inverse Permutation Table (*Z*-axis Rotation)							
0	49	62	15	4	33	58	31
8	17	54	47	12	1	50	63
16	53	46	11	20	37	42	27
24	21	38	43	28	5	34	59
32	57	30	7	36	41	26	23
40	25	22	39	44	9	18	55
48	61	14	3	52	45	10	19
56	29	6	35	60	13	2	51

Output(1) = **Input(1)** → Inverse P-table:
0011111001100101111011010011110001011100010000110111000111100101
(*3E65ED3C5C4371E5*)

2) **Process 2**: **Add Round Key (3D Bit Rotation)**

Input(2_a) = **Output(1)**:
0011111001100101111011010011110001011100010000110111000111100101 (*3E65ED3C5C4371E5*)

Input(2_b) = ***SubKey*$_{20}$**:
1101000011010000100001110010011101100101000100001010001010011000 (*D0D087276510A298*)

i) **Step 1**: **Input(2_a)** XOR **Input(2_b)**.

Output(2):
1110111010110101011010100001101100111001010100111101001101111101
(*EEB56A1B3953D37D*)

3) Process 3: 3D Bit Rotation (*X*-axis)

Input(3) = **Output(2)**:
1110111010110101011010100001101100111001010100111101001101111101 (*EEB56A1B3953D37D*)

i) Step 1: Permute the bits of **Input(3)** according to the position from the 3D Bit Rotation Inverse Permutation Table (*X*-axis) values as shown in Table 2.

Table 2: 3D Bit Rotation Inverse Permutation Table (X-axis)

Inverse Permutation Table (*X*-axis Rotation)							
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
19	23	27	31	18	22	26	30
17	21	25	29	16	20	24	28
47	46	45	44	43	42	41	40
39	38	37	36	35	34	33	32
60	56	52	48	61	57	53	49
62	58	54	50	63	59	55	51

Output(3) = **Input(3)** → Inverse P-table:
1110111010110101001111011000010111001010100111001001110101101111
(*EEB53D85CA9C9D6F*)

4) Process 4: Cipher State Conversion

Input(4) = **Output(3):** 1110111010110101001111011000010111001010100111001001110101101111
(*EEB53D85CA9C9D6F*)

i) Step 1: Divide **Input(4)** into 4 blocks.

Output(4_a) = 1st 16 bits of **Input(4)**: 1110111010110101 (*EEB5*) : ***RowKey*$_3$**
Output(4_b) = 2nd 16 bits of **Input(4)**: 0011110110000101 (*3D85*) : ***RowKey*$_2$**
Output(4_c) = 3rd 16 bits of **Input(4)**: 1100101010011100 (*CA9C*) : ***RowKey*$_1$**
Output(4_d) = 4th 16 bits of **Input(4)**: 1001110101101111 (*9D6F*) : ***RowKey*$_0$**

5) Process 5: Sub Column

Input(5_a) = ***RowKey*$_3$**: 1110111010110101 (*EEB5*)
Input(5_b) = ***RowKey*$_2$**: 0011110110000101 (*3D85*)
Input(5_c) = ***RowKey*$_1$**: 1100101010011100 (*CA9C*)
Input(5_d) = ***RowKey*$_0$**: 1001110101101111 (*9D6F*)

i) Step 1: Append 1 bit each from **Input(5_a)**, **Input(5_b)**, **Input(5_c)**, and **Input(5_d)** to obtain 4 bits output. Then, convert the output to a decimal value. Repeat for 16 times.

Output(5_{e1}) = 1011: 11
Output(5_{e2}) = 1010: 10
Output(5_{e3}) = 1100: 12
Output(5_{e4}) = 0101: 5
Output(5_{e5}) = 1111: 15
Output(5_{e6}) = 1101: 13
Output(5_{e7}) = 1010: 10
Output(5_{e8}) = 0101: 5
Output(5_{e9}) = 1110: 14
Output(5_{e10}) = 0001: 1
Output(5_{e11}) = 1001: 9
Output(5_{e12}) = 1010: 10
Output(5_{e13}) = 0011: 3
Output(5_{e14}) = 1111: 15
Output(5_{e15}) = 0001: 1
Output(5_{e16}) = 1101: 13

Step 2: Replace **Output(5_{e1})** to **Output(5_{e16})** with the PRESENT Inverse S-box value as shown in Table 3. Then, convert the output to binary.

Table 3: PRESENT Inverse S-box

X (Input)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(*x*) (Output)	5	E	F	8	C	1	2	D	B	4	6	3	0	7	9	A

Output(5_{f1}) = Output(5_{e1}) → Inverse S-box = 3: 0011
Output(5_{f2}) = Output(5_{e2}) → Inverse S-box = 6: 0110
Output(5_{f3}) = Output(5_{e3}) → Inverse S-box = 0: 0000
Output(5_{f4}) = Output(5_{e4}) → Inverse S-box = 1: 0001
Output(5_{f5}) = Output(5_{e5}) → Inverse S-box = 10: 1010
Output(5_{f6}) = Output(5_{e6}) → Inverse S-box = 7: 0111
Output(5_{f7}) = Output(5_{e7}) → Inverse S-box = 6: 0110
Output(5_{f8}) = Output(5_{e8}) → Inverse S-box = 1: 0001
Output(5_{f9}) = Output(5_{e9}) → Inverse S-box = 9: 1001
Output(5_{f10}) = Output(5_{e10}) → Inverse S-box = 14: 1110
Output(5_{f11}) = Output(5_{e11}) → Inverse S-box = 4: 0100
Output(5_{f12}) = Output(5_{e12}) → Inverse S-box = 6: 0110
Output(5_{f13}) = Output(5_{e13}) → Inverse S-box = 8: 1000
Output(5_{f14}) = Output(5_{e14}) → Inverse S-box = 10: 1010
Output(5_{f15}) = Output(5_{e15}) → Inverse S-box = 14: 1110
Output(5_{f16}) = Output(5_{e16}) → Inverse S-box = 7: 0111

ii) **Step 3**: Append 1 bit each from **Output(5_{f1})** to **Output(5_{f16})** to obtain 16 bits output. Repeat for 4 times. Update the new ***RowKeys***.

Output(5_g): 0000100011001110 (*08CE*): ***RowKey*$_3$**
Output(5_h): 0100011001110011 (*4673*): ***RowKey*$_2$**
Output(5_i): 1100111001010111 (*CE57*): ***RowKey*$_1$**
Output(5_j): 1001010110000001 (*9581*): ***RowKey*$_0$**

iii) **Step 4**: Append **Output(5_g)**, **Output(5_h)**, **Output(5_i)**, and **Output(5_j)** to obtain 64 bits output to obtain the ***Plaintext*** for current round.

Output(5_a) = 0000100011001110010001100111001111001110010101111001010110000001
(*08CE4673CE579581*): ***Plaintext*$_1$**
* **Output(4_b)** in this example is the generated ***Plaintext*$_1$**.

6) **Process 6**: Generate the remaining ***Plaintext*** (***Plaintext*$_2$** to ***Plaintext*$_{19}$**)

i) **Step 1:** Repeat **Process 1**, **Process 2**, **Process 3**, **Process 4**, and **Process 5** for another 18 times to generate ***Plaintext*$_2$** to ***Plaintext*$_{19}$**.

a) **Process 1** (3D Bit Rotation *Z*-axis)
Input: ***Plaintext*** from **Process 5** (previous round Sub Column).
Output: **Output(1)**.

b) **Process 2** (Add Round Key 3D Bit Rotation)
Input: ***SubKey*** and **Output(1)** from **Process 1** (current round 3D Bit Rotation *Z*-axis).
Output: **Output(2)**.

c) **Process 3** (3D Bit Rotation *X*-axis)
Input: **Output(2)** from **Process 2** (current round Add Round Key 3D Bit Rotation).
Output: **Output(3)**.

d) **Process 4** (Cipher State Conversion)
Input: **Output(3)** from **Process 3** (current round 3D Bit Rotation *X*-axis).
Output: Updated ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*$_0$**.

e) **Process 5** (Sub Column)
Input: ***RowKey*$_3$**, ***RowKey*$_2$**, ***RowKey*$_1$**, and ***RowKey*0** from **Process 4** (current round Cipher State Conversion).
Output: ***Plaintext***.

ii) **Step 2**: Obtain all rounds ***Plaintext*** (***Plaintext*$_1$** to ***Plaintext*$_{19}$**).

Output(6_A) = 0000100011001110010001100111001111001110010101111001010110000001
(*08CE4673CE579581*): ***Plaintext*$_1$**

Output(6_B) = 1000100100110101010111001100101111011001000011101000101000100101
(*89355CCBD90E8A25*): ***Plaintext*$_2$**

Output(6_C) = 0101011110111010101110001000010001000010011101101110110111000101
(*57BAB8844276EDC5*): ***Plaintext*$_3$**

Output(6_D) = 1111101111010111101011101010111101101001100010110011111111000101
(*FBD7AEAF698B3FC5*): ***Plaintext*$_4$**

Output(6_E) = 1111100011011110110110010111000000111011001111001101101100001100
(*F8DED9703B3CDB0C*): ***Plaintext*$_5$**

Output(6_F) = 0011011110101101110101101000111001100100100001000010001001000001
(*37ADD68E64842241*): ***Plaintext*$_6$**

Output(6_G) = 1101010100000100001011100100100001110010101011000001011110110011
(*D5042E4872AC17B3*): ***Plaintext*$_7$**

Output(6_H) = 0100001010100110011101011010110011010100110111000001001101111110
(*454CEB59A9B826FE*): ***Plaintext*$_8$**

Output(6_I) = 0111101001100110110101010111100001110001000011100001111101111011
(*7A66D578710E1F7B*): ***Plaintext*$_9$**

Output(6_J) = 1110000000011011110010000010101001110011100111001111001010001000
(*E01BC82A739E7A88*): ***Plaintext*$_{10}$**

Output(6_K) = 0101010100010110101110110001111101000110111011000110001001010100
(*5516BB1F46EC6254*): ***Plaintext*$_{11}$**

Output(6_L) = 1010000011000010011110000100011111110001110100001000001111110011
(*A0C27847F1D083F3*): ***Plaintext*$_{12}$**

Output(6_M) = 0101011110000011011111011101110100001001111000010100101101000000
(*57837DDD09E14B40*): ***Plaintext*$_{13}$**

Output(6_N) = 0101101001111110000110101111101010010011010001101101100000011011
(*5A7E1AFA9346D81B*): ***Plaintext*$_{14}$**

Output(6_O) = 1000011011000000011111110001010101111110101011011001001000000100
(*86C07E2AFD5B2404*): ***Plaintext*$_{15}$**

Output(6_P) = 1111001101111110001001100011101001001001101001000101000001111111
(*F37E263A49A4507F*): ***Plaintext*$_{16}$**

Output(6_Q) = 1011110000011101101010001101011111101000000100101001010100010010
(*BC1DA8D7E8129512*): ***Plaintext*$_{17}$**

Output(6_R) = 110001101111111000001001100001011100010111110000000010110110111 1
(*C6FE0985C5F00B6F*): ***Plaintext*$_{18}$**

Output(6_S) = 010000101000001010000010001100100101101001100010010010010100000 10
(*428282325A624A82*): ***Plaintext*$_{19}$**

7) <u>Process 7</u>: <u>Add Round Key</u>

Input(7_a) = ***Plaintext*$_{19}$**:
010000101000001010000010001100100101101001100010010010010100000 10 (*428282325A624A82*)

Input(7_b) = ***SubKey*$_0$**:
010000101000001010000010001100100101101001100010010010010100000 10 (*428282325A624A82*)

i) <u>Step 1</u>: Input(7_a) XOR **Input(7_b).**

Output(7_a): 000000000000000000000000000000000000000000000000000000000000000 00
(*0000000000000000*)

ii) <u>Step 2</u>: Obtain ***Plaintext*.**

Output(7) = **Output(7_a)**
= 000000000000000000000000000000000000000000000000000000000000000 00
(*0000000000000000*): ***Plaintext***

APPENDIX D

Source Code of LAO-3D Lightweight Block Cipher (Encryption)

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <math.h>
#include <bitset>
#include <sstream>
using namespace std;

char ArrayS[256];
int OutputI,tempInt,kira,S1,X1,Y1,Z1;
string OutputS,tempStr,plaintext,CiphertextHex,CiphertextBin,
   OutputCipherBin,OutputCipherHex,tempB,KeyChar,MessageChar,
   CiphertextBinAll;
string RowKey[8],OutputArrayS[32],SubKey[30],SubColumn_In[16],
   SubColumn_Out[16],RowKeyNew[8],Row[8],Ciphertext[30],Partition[100],
   Hx[100],MsgBlock[400];

// BOX //
int S_Box[16] = {12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2};
int Rot_X_axis[64]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,28,24,20,
   16,29,25,21,17,30,26,22,18,31,27,23,19,47,46,45,44,43,42,41,40,39,38,
   37,36,35,34,33,32,51,55,59,63,50,54,58,62,49,53,57,61,48,52,56,60};
int Rot_Z_axis[64]={0,13,62,51,4,29,58,35,8,45,54,19,12,61,50,3,16,9,
   46,55,20,25,42,39,24,41,38,23,28,57,34,7,32,5,30,59,36,21,26,43,40,
   37,22,27,44,53,18,11,48,1,14,63,52,17,10,47,56,33,6,31,60,49,2,15};
string Nonce = "01000001010000100100010001010101010011000100000101001
   10001001001010001100101101001000001010010110100000101010010010010101
   000001";

// FUNCTION //
void Divide_String_To_Block_Reverse (string InputS, int InputI)
{
   for(int i=0;i<InputS.length()/InputI;i++){OutputArrayS[i]="";
   for(int j=0;j<InputI;j++){OutputArrayS[i]+=InputS.at
   (i*InputI+(InputI-j-1));}}
}

string XOR(string InputS1, string InputS2)
{
   OutputS="";for(int i=0;i<InputS1.length();i++)
   {if(InputS1.at(i)!=InputS2.at(i)){OutputS += "1";}
   else{OutputS += "0";}}return(OutputS);
}

```

```
33 int Binary_Decimal(string InputS, int InputI)
34 {
35    OutputI=0;for(int i=0;i<InputI;i++)
      {OutputI=OutputI+(InputS.at(i)-48)*
      (int)pow(2.0,InputI-1-i);}return(OutputI);
36 }
37
38 string Shift_Left(string InputS, int InputI)
39 {
40    OutputS="";for(int i=0;i<InputS.length();i++)
      {OutputS+=InputS.at((InputS.length()+InputI+i)%
      InputS.length());}return(OutputS);
41 }
42
43 string Decimal_To_Binary(unsigned int InputI1, int InputI2)
44 {
45    kira=0;OutputS="";
46 ulang:
47    kira++;ArrayS[kira]=InputI1%2+48;
      if(InputI1/2!=0){InputI1=InputI1/2;goto ulang;}
      for(int i=0;i<InputI2-kira;i++){OutputS=OutputS+'0';}
48    for(int i=kira;i>0;i--){OutputS=OutputS+ArrayS[i];}
      return(OutputS);
49 }
50
51 void Divide_String_To_Block (string InputS, int InputI)
52 {
53    for(int i=0;i<InputS.length()/InputI;i++)
      {OutputArrayS[i]="";for(int j=0;j<InputI;j++)
      {OutputArrayS[i]+=InputS.at(i*InputI+j);}}
54 }
55
56 string Bin2Hex(string InputS)
57 {
58    OutputS="";
59    if(InputS.length()%4 == 0)
60    {
61      for(int i=0;i<InputS.length()/4;i++)
62      {
63          Partition[i]="";for(int j=0;j<4;j++)
            {Partition[i]=Partition[i]+InputS.at(i*4+j);}
64          if(Partition[i]=="0000"){Hx[i]='0';}
            if(Partition[i]=="0001"){Hx[i]='1';}
            if(Partition[i]=="0010"){Hx[i]='2';}
            if(Partition[i]=="0011"){Hx[i]='3';}
65          if(Partition[i]=="0100"){Hx[i]='4';}
            if(Partition[i]=="0101"){Hx[i]='5';}
            if(Partition[i]=="0110"){Hx[i]='6';}
            if(Partition[i]=="0111"){Hx[i]='7';}
66          if(Partition[i]=="1000"){Hx[i]='8';}
            if(Partition[i]=="1001"){Hx[i]='9';}
            if(Partition[i]=="1010"){Hx[i]='a';}
            if(Partition[i]=="1011"){Hx[i]='b';}
67          if(Partition[i]=="1100"){Hx[i]='c';}
            if(Partition[i]=="1101"){Hx[i]='d';}
            if(Partition[i]=="1110"){Hx[i]='e';}
```

```
            if(Partition[i]=="1111"){Hx[i]='f';}
68          OutputS=OutputS+Hx[i];
69       }
70    }
71    return(OutputS);
72 }
73
74 string char_bin_Key(string InputS)
75 {
76    if(InputS.length()<16){X1=16-InputS.length();
      for(int i=0;i<X1;i++){InputS=InputS+char(32);}}
77    if(InputS.length()>16){tempStr="";
      for(int i=0;i<16;i++){tempStr=tempStr+InputS.at(i);}
      InputS=tempStr;}
78    OutputS="";for (size_t i = 0; i < InputS.size(); ++i)
      {OutputS=OutputS+bitset<8>(InputS[i]).to_string();}
      return(OutputS);
79 }
80
81 void char_bin_Message(string InputS)
82 {
83    OutputCipherBin="";Y1=InputS.length()/8;
84    if(InputS.length()%8 != 0){Y1=Y1+1;S1=InputS.length()%8;
      Z1=Y1*8-InputS.length();for(int i=Y1-1;i<Y1;i++)
      {for(int j=0;j<Z1;j++){InputS=InputS+char(32);}}}
85
86    for(int k=0;k<Y1;k++)
87    {
88       MsgBlock[k]="";tempStr="";for(int j=0;j<8;j++)
          {tempStr=tempStr+InputS.at(k*8+j);}
          for (size_t i = 0; i < 8; ++i)
          {MsgBlock[k]=MsgBlock[k]+bitset<8>(tempStr[i]).to_string();}
89       OutputCipherBin=OutputCipherBin+MsgBlock[k];
90    }
91 }
92
93 ////////////////////////////////////////////////////////////////
94 // ALGORITHM CODE //
95 ////////////////////////////////////////////////////////////////
96 // KEY SCHEDULE //
97 //==============================================================//
98
99 void Row_Key(string masterkey, string Nonce)
100 {
101   Divide_String_To_Block_Reverse (XOR(masterkey, Nonce), 32);
      for(int i=0;i<4;i++){RowKey[i]=OutputArrayS[3-i];}
102 }
103
104 void Sub_Key(int InputI)
105 {
106   SubKey[InputI]="";for(int j=0;j<4;j++)
      {for(int i=16;i<32;i++){tempStr=RowKey[3-j].at(i);
      SubKey[InputI]+=tempStr;}}Bin2Hex(SubKey[InputI]);
107 }
108
```

```
109 void Key_Sub_Column()
110 {
111   for(int k=0;k<8;k++)
112   {
113      SubColumn_In[k]="";tempStr="";
       for(int j=0;j<4;j++){tempStr+=RowKey[3-j].at(32-8+k);}
       SubColumn_In[k]=tempStr;Binary_Decimal(SubColumn_In[k],4);
       tempInt=S_Box[OutputI];
114    SubColumn_Out[k]=Decimal_To_Binary(tempInt, 4);
       for(int j=0;j<4;j++){tempStr+=RowKey[j].at(32-8+k);
       RowKey[3-j].at(32-8+k)=SubColumn_Out[k].at(j);}
115   }
116 }
117
118 void Row_transformation()
119 {
120   Shift_Left(RowKey[0],8);RowKeyNew[0]=OutputS;
     XOR(RowKeyNew[0],RowKey[1]);RowKeyNew[0]=OutputS;
     RowKeyNew[1]=RowKey[2];Shift_Left(RowKey[2],16);
     RowKeyNew[2]=OutputS;
121  XOR(RowKeyNew[2],RowKey[3]);RowKeyNew[2]=OutputS;
     RowKeyNew[3]=RowKey[0];
     for(int i=0;i<4;i++){RowKey[i]=RowKeyNew[i];}
122 }
123
124 void keyschedule(string masterkey, string Nonce)
125 {
126   Row_Key(masterkey, Nonce);
     for(int Round=0;Round<21;Round++){Sub_Key(Round);
     if(Round==20){break;}Key_Sub_Column();Row_transformation();}
127 }
128
129 //============================================================//
130 // ENCRYPTION //
131 //============================================================//
132
133 string AddRoundkey(string InputS, int InputI)
134 {
135   XOR(InputS, SubKey[InputI]);return(OutputS);
136 }
137
138 void Sub_Column()
139 {
140   Divide_String_To_Block (OutputS, 16);
     for(int i=0; i<4; i++){Row[3-i]=OutputArrayS[i];}
141
142   for(int k=0;k<16;k++)
143   {
144      tempStr="";for(int j=0;j<4;j++){tempStr+=Row[3-j].at(k);}
       SubColumn_In[k]=tempStr;Binary_Decimal(SubColumn_In[k],4);
       tempInt=S_Box[OutputI];
145    SubColumn_Out[k]=Decimal_To_Binary(tempInt, 4);
       for(int j=0;j<4;j++){tempStr+=Row[j].at(k);
       Row[3-j].at(k)=SubColumn_Out[k].at(j);}
146   }
```

```
147   OutputS="";for(int i=0;i<4;i++){OutputS+=Row[3-i];}
148 }
149
150 void DoubleRotation3D(int InputI)
151 {
152   tempStr="";for(int i=0;i<64;i++)
      {tempStr+=OutputS.at(Rot_X_axis[i]);}AddRoundkey(tempStr,InputI);
153   tempStr="";for(int i=0;i<64;i++)
      {tempStr+=OutputS.at(Rot_Z_axis[i]);}OutputS=tempStr;
154 }
155
156 string encryption(string plaintext)
157 {
158   AddRoundkey(plaintext, 0);for(int Round=1;Round<21;Round++)
      {Sub_Column();DoubleRotation3D(Round);Ciphertext[Round]=OutputS;}
      CiphertextBin=Ciphertext[20];
159   Bin2Hex(CiphertextBin);CiphertextHex=OutputS;
      return(CiphertextHex);
160 }
161
162 //============================================================//
163
164 int main()
165 {
166   tempB=""; OutputCipherHex=""; KeyChar="";MessageChar="";
      cout<<"Encryption"<<endl<<"==========="<<endl<<endl;
167   cout<<"Enter Key: "; getline(cin,KeyChar);char_bin_Key(KeyChar);
      keyschedule(OutputS, Nonce);
168   cout<<endl<<"Enter Message: "<<endl;getline(cin,MessageChar);
      char_bin_Message(MessageChar);
169   if(MessageChar.length()>2040)
      {cout<<endl<<"Message Length:"<<MessageChar.length()<<endl<<endl;
      cout<<"(Exceeded message length limit)"<<endl<<endl; return 0;}
170   for(int ab=0;ab<Y1;ab++){encryption(MsgBlock[ab]);
      {OutputCipherHex=OutputCipherHex+CiphertextHex;}
      tempB=tempB+CiphertextBin;}
171   cout<<endl<<"Ciphertext: "<<endl<<OutputCipherHex<<endl<<endl;
      CiphertextBinAll=tempB;
172
173   return 0;
174 }
```

APPENDIX E

Source Code of LAO-3D Lightweight Block Cipher (Decryption)

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <math.h>
#include <bitset>
#include <sstream>
using namespace std;

char ArrayS[256];
int OutputI,tempInt,kira,S1,X1,Y1,Z1;
string OutputS,tempStr,PlaintextBin,PlaintextChar,OutputPlain,
   ciphertext,OutputCipherBin,tempB,KeyChar,MessageChar;
string RowKey[8],OutputArrayS[32],SubKey[30],SubColumn_In[16],
   SubColumn_Out[16],RowKeyNew[8],Row[8],Plaintext[30],Partition[100],
   Hx[100],MsgBlock[400];

// BOX //
int S_Box[16] = {12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2};
int Inverse_S_Box[16] = {5,14,15,8,12,1,2,13,11,4,6,3,0,7,9,10};
int Inverse_Rot_X_axis[64]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,19,
   23,27,31,18,22,26,30,17,21,25,29,16,20,24,28,47,46,45,44,43,42,41,40,
   39,38,37,36,35,34,33,32,60,56,52,48,61,57,53,49,62,58,54,50,63,59,55,
   51};
int Inverse_Rot_Z_axis[64]={0,49,62,15,4,33,58,31,8,17,54,47,12,1,50,
   63,16,53,46,11,20,37,42,27,24,21,38,43,28,5,34,59,32,57,30,7,36,41,
   26,23,40,25,22,39,44,9,18,55,48,61,14,3,52,45,10,19,56,29,6,35,60,13,
   2,51};
string Nonce = "01000001010000100100001000101010101001100010000010100
   1000100100101000110010110100100000101001011010000010101001001001001010
   000001";

////////////////////////////////////////////////////////////
// FUNCTION //
////////////////////////////////////////////////////////////
void Divide_String_To_Block_Reverse (string InputS, int InputI)
{
   for(int i=0;i<InputS.length()/InputI;i++)
   {OutputArrayS[i]="";for(int j=0;j<InputI;j++)
   {OutputArrayS[i]+=InputS.at(i*InputI+(InputI-j-1));}}
}

```

```
31 string XOR(string InputS1, string InputS2)
32 {
33    OutputS="";for(int i=0;i<InputS1.length();i++)
      {if(InputS1.at(i)!=InputS2.at(i)){OutputS += "1";}
      else{OutputS += "0";}}return(OutputS);
34 }
35
36 int Binary_Decimal(string InputS, int InputI)
37 {
38    OutputI=0;for(int i=0;i<InputI;i++)
      {OutputI=OutputI+(InputS.at(i)-48)*(int)pow(2.0,InputI-1-i);}
      return(OutputI);
39 }
40
41 string Shift_Left(string InputS, int InputI)
42 {
43    OutputS="";for(int i=0;i<InputS.length();i++)
      {OutputS+=InputS.at((InputS.length()+InputI+i)%InputS.length());}
      return(OutputS);
44 }
45
46 string Decimal_To_Binary(unsigned int InputI1, int InputI2)
47 {
48    kira=0;OutputS="";
49 ulang:
50    kira++;ArrayS[kira]=InputI1%2+48; if(InputI1/2!=0)
      {InputI1=InputI1/2;goto ulang;}
      for(int i=0;i<InputI2-kira;i++){OutputS=OutputS+'0';}
51    for(int i=kira;i>0;i--){OutputS=OutputS+ArrayS[i];}
      return(OutputS);
52 }
53
54 void Divide_String_To_Block (string InputS, int InputI)
55 {
56    for(int i=0;i<InputS.length()/InputI;i++)
      {OutputArrayS[i]="";for(int j=0;j<InputI;j++)
      {OutputArrayS[i]+=InputS.at(i*InputI+j);}}
57 }
58
59 string Bin2Hex(string InputS)
60 {
61    OutputS="";
62    if(InputS.length()%4 == 0)
63    {
64       for(int i=0;i<InputS.length()/4;i++)
65       {
66          Partition[i]="";for(int j=0;j<4;j++)
          {Partition[i]=Partition[i]+InputS.at(i*4+j);}
67          if(Partition[i]=="0000"){Hx[i]='0';}
          if(Partition[i]=="0001"){Hx[i]='1';}
          if(Partition[i]=="0010"){Hx[i]='2';}
          if(Partition[i]=="0011"){Hx[i]='3';}
68          if(Partition[i]=="0100"){Hx[i]='4';}
          if(Partition[i]=="0101"){Hx[i]='5';}
          if(Partition[i]=="0110"){Hx[i]='6';}
          if(Partition[i]=="0111"){Hx[i]='7';}
```

```
69          if(Partition[i]=="1000"){Hx[i]='8';}
            if(Partition[i]=="1001"){Hx[i]='9';}
            if(Partition[i]=="1010"){Hx[i]='a';}
            if(Partition[i]=="1011"){Hx[i]='b';}
70          if(Partition[i]=="1100"){Hx[i]='c';}
            if(Partition[i]=="1101"){Hx[i]='d';}
            if(Partition[i]=="1110"){Hx[i]='e';}
            if(Partition[i]=="1111"){Hx[i]='f';}
71          OutputS=OutputS+Hx[i];
72       }
73    }
74    return(OutputS);
75 }
76
77 string Hex2Bin_Message(string InputS)
78 {
79    OutputS="";tempB="";Y1=InputS.length()/16;
80    for(int i=0;i<InputS.length();i++)
81    {
82       tempStr="";if(InputS.at(i)=='0'){tempStr="0000";}
          if(InputS.at(i)=='1'){tempStr="0001";}
          if(InputS.at(i)=='2'){tempStr="0010";}
          if(InputS.at(i)=='3'){tempStr="0011";}
83        if(InputS.at(i)=='4'){tempStr="0100";}
          if(InputS.at(i)=='5'){tempStr="0101";}
          if(InputS.at(i)=='6'){tempStr="0110";}
          if(InputS.at(i)=='7'){tempStr="0111";}
84        if(InputS.at(i)=='8'){tempStr="1000";}
          if(InputS.at(i)=='9'){tempStr="1001";}
          if(InputS.at(i)=='A' || InputS.at(i)=='a'){tempStr="1010";}
85        if(InputS.at(i)=='B' || InputS.at(i)=='b'){tempStr="1011";}
          if(InputS.at(i)=='C' || InputS.at(i)=='c'){tempStr="1100";}
86        if(InputS.at(i)=='D' || InputS.at(i)=='d'){tempStr="1101";}
          if(InputS.at(i)=='E' || InputS.at(i)=='e'){tempStr="1110";}
87        if(InputS.at(i)=='F' || InputS.at(i)=='f'){tempStr="1111";}
88        tempB=tempB+tempStr;
89    }
90    for(int k=0;k<Y1;k++){MsgBlock[k]="";for(int j=0;j<64;j++)
      {MsgBlock[k]=MsgBlock[k]+tempB.at(k*64+j);}
      OutputS=OutputS+MsgBlock[k];}return(OutputS);
91 }
92
93 string char_bin_Key(string InputS)
94 {
95    if(InputS.length()<16){X1=16-InputS.length();
      for(int i=0;i<X1;i++){InputS=InputS+char(32);}}
96    if(InputS.length()>16){tempStr="";for(int i=0;i<16;i++)
      {tempStr=tempStr+InputS.at(i);}InputS=tempStr;}
97    OutputS="";for (size_t i = 0; i < InputS.size(); ++i)
      {OutputS=OutputS+bitset<8>(InputS[i]).to_string();}
      return(OutputS);
98 }
99
```

```
100 void char_bin_Message(string InputS)
101 {
102   OutputCipherBin="";Y1=InputS.length()/8;
103   if(InputS.length()%8 != 0){Y1=Y1+1;S1=InputS.length()%8;
      Z1=Y1*8-InputS.length();for(int i=Y1-1;i<Y1;i++)
      {for(int j=0;j<Z1;j++){InputS=InputS+char(32);}}}
104
105   for(int k=0;k<Y1;k++)
106   {
107      MsgBlock[k]="";tempStr="";for(int j=0;j<8;j++)
         {tempStr=tempStr+InputS.at(k*8+j);}
         for (size_t i = 0; i < 8; ++i)
         {MsgBlock[k]=MsgBlock[k]+bitset<8>(tempStr[i]).to_string();}
108      OutputCipherBin=OutputCipherBin+MsgBlock[k];
109   }
110 }
111
112 string bin_char_Message(string InputS)
113 {
114   OutputS="";stringstream sstream(InputS);
      while(sstream.good()){bitset<8>bits;sstream>>bits;
      OutputS=OutputS+char(bits.to_ulong());}return(OutputS);
115 }
116
117 ////////////////////////////////////////////////////////////////
118 // ALGORITHM CODE //
119 ////////////////////////////////////////////////////////////////
120 //================================================================//
121 // KEY SCHEDULE //
122
123 void Row_Key(string masterkey, string Nonce)
124 {
125   Divide_String_To_Block_Reverse (XOR(masterkey, Nonce), 32);
      for(int i=0;i<4;i++){RowKey[i]=OutputArrayS[3-i];}
126 }
127
128 void Sub_Key(int InputI)
129 {
130   SubKey[InputI]="";for(int j=0;j<4;j++){for(int i=16;i<32;i++)
      {tempStr=RowKey[3-j].at(i);SubKey[InputI]+=tempStr;}}
      Bin2Hex(SubKey[InputI]);
131 }
132
133 void Key_Sub_Column()
134 {
135   for(int k=0;k<8;k++)
136   {
137      SubColumn_In[k]="";tempStr="";for(int j=0;j<4;j++)
         {tempStr+=RowKey[3-j].at(32-8+k);}SubColumn_In[k]=tempStr;
         Binary_Decimal(SubColumn_In[k],4);tempInt=S_Box[OutputI];
138      SubColumn_Out[k]=Decimal_To_Binary(tempInt, 4);
         for(int j=0;j<4;j++){tempStr+=RowKey[j].at(32-8+k);
         RowKey[3-j].at(32-8+k)=SubColumn_Out[k].at(j);}
139   }
140 }
141
```

```
142 void Row_transformation()
143 {
144   Shift_Left(RowKey[0],8);RowKeyNew[0]=OutputS;
      XOR(RowKeyNew[0],RowKey[1]);
      RowKeyNew[0]=OutputS;RowKeyNew[1]=RowKey[2];
      Shift_Left(RowKey[2],16);RowKeyNew[2]=OutputS;
145   XOR(RowKeyNew[2],RowKey[3]);RowKeyNew[2]=OutputS;
      RowKeyNew[3]=RowKey[0];
      for(int i=0;i<4;i++){RowKey[i]=RowKeyNew[i];}
146 }
147
148 void keyschedule(string masterkey, string Nonce)
149 {
150   Row_Key(masterkey, Nonce);for(int Round=0;Round<21;Round++)
      {Sub_Key(Round);if(Round==20){break;}Key_Sub_Column();
      Row_transformation();}
151 }
152 //============================================================//
153 // DECRYPTION //
154 //============================================================//
155 string AddRoundkey(string InputS, int InputI)
156 {
157   XOR(InputS, SubKey[InputI]);return(OutputS);
158 }
159
160 void Decrypt_DoubleRotation3D(int Int)
161 {
162   tempStr="";for(int i=0;i<64;i++)
      {tempStr+=OutputS.at(Inverse_Rot_Z_axis[i]);}
      AddRoundkey(tempStr, Int);
163   tempStr="";for(int i=0;i<64;i++)
      {tempStr+=OutputS.at(Inverse_Rot_X_axis[i]);}OutputS=tempStr;
164 }
165
166 void Decrypt_Sub_Column()
167 {
168   Divide_String_To_Block (OutputS, 16);for(int i=0; i<4; i++)
      {Row[3-i]=OutputArrayS[i];}
169
170   for(int k=0;k<16;k++)
171   {
172      tempStr="";for(int j=0;j<4;j++){tempStr+=Row[3-j].at(k);}
       SubColumn_In[k]=tempStr;Binary_Decimal(SubColumn_In[k],4);
       tempInt=Inverse_S_Box[OutputI];
173      SubColumn_Out[k]=Decimal_To_Binary(tempInt, 4);
       for(int j=0;j<4;j++){tempStr+=Row[j].at(k);
       Row[3-j].at(k)=SubColumn_Out[k].at(j);}
174   }
175   OutputS="";for(int i=0;i<4;i++){OutputS+=Row[3-i];}
176 }
177
```

```
178 string decryption(string ciphertext)
179 {
180   tempB="";OutputS=ciphertext;for(int Round=1;Round<21;Round++)
      {Decrypt_DoubleRotation3D(21-Round);Decrypt_Sub_Column();
      Plaintext[Round]=OutputS;}
181   AddRoundkey(Plaintext[20],0);PlaintextBin=OutputS;
      bin_char_Message(PlaintextBin);
      PlaintextChar=OutputS;return(PlaintextChar);
182 }
183
184 //============================================================//
185
186 int main()
187 {
188   KeyChar=""; MessageChar="";OutputPlain="";
      cout<<"Decryption"<<endl<<"=========="<<endl<<endl;
189   cout<<"Enter Key: ";getline(cin,KeyChar);char_bin_Key(KeyChar);
      keyschedule(OutputS, Nonce);
190   cout<<endl<<"Enter Ciphertext: "<<endl;getline(cin,MessageChar);
      Hex2Bin_Message(MessageChar);OutputCipherBin=OutputS;
191   if(MessageChar.length()>4080)
      {cout<<endl<<"Ciphertext Length:"
      <<MessageChar.length()<<endl<<endl;
      cout<<"(Exceeded ciphertext length limit)"<<endl<<endl; return 0;}
192   for(int ab=0;ab<Y1;ab++){decryption(MsgBlock[ab]);
      for(int bc=0;bc<8;bc++)
      {OutputPlain=OutputPlain+PlaintextChar.at(bc);}}
193   cout<<endl<<"Plaintext:"<<endl<<OutputPlain<<endl<<endl;
      char_bin_Message(OutputPlain);
194
195   return 0;
196 }
```

APPENDIX F

Source Code of LAO-3D Lightweight Block Cipher Sample Generation for Randomness Tests Using Nine Data Categories

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <math.h>
using namespace std;

 string plaintext,ciphertext,masterkey,tempStr,OutputS,FinalCipher,
 OutputCipher;
string OutputArrayS[32],Row[4],RowKey[4],RowKeyNew[4],SubKey[26],
 SubColumn_In[8],SubColumn_Out[8],Ciphertext[30];
int OutputI,tempInt,kira,Round,ROUNDS;

int S_Box[16] = {12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2};

int Rot_X_axis[64]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,28,24,20,
 16,29,25,21,17,30,26,22,18,31,27,23,19,47,46,45,44,43,42,41,40,39,38,
 37,36,35,34,33,32,51,55,59,63,50,54,58,62,49,53,57,61,48,52,56,60};

int Rot_Z_axis[64]={0,13,62,51,4,29,58,35,8,45,54,19,12,61,50,3,16,9,
 46,55,20,25,42,39,24,41,38,23,28,57,34,7,32,5,30,59,36,21,26,43,40,
 37,22,27,44,53,18,11,48,1,14,63,52,17,10,47,56,33,6,31,60,49,2,15};

ofstream myfile1;

////////////////////////////////////////////////////////////
//              PARAMETERS FOR DATA CATEGORIES            //
////////////////////////////////////////////////////////////
char text,semen[256],ArrayS[256];
int para,KeyBlockSize,PlaintextBlockSize,DataCategory,mula,akhir;
string iv,plainIV,asal,KeyName,TextName,line,line2,inputkey,
 samplekey,sampletext,inputtext,cipher1,cipher2,OutputFunction;
string key1bit[256],key2bit[9999],text1bit[128],main2bit[9999],
 CIPHERtext[512],InputTeks[9999],InputKunci[9999],flippedkey[512],
 flippedtext[512];

////////////////////////////////////////////////////////////////
//                        OTHERS FUNCTIONS                    //
////////////////////////////////////////////////////////////////
```

```
string Function_HexToBin (string Str1)
{
   OutputFunction="";
   for (int i=0;i<Str1.length();++i)
   {
      switch (Str1 [i])
      {
         case '0': OutputFunction.append ("0000"); break;
         case '1': OutputFunction.append ("0001"); break;
         case '2': OutputFunction.append ("0010"); break;
         case '3': OutputFunction.append ("0011"); break;
         case '4': OutputFunction.append ("0100"); break;
         case '5': OutputFunction.append ("0101"); break;
         case '6': OutputFunction.append ("0110"); break;
         case '7': OutputFunction.append ("0111"); break;
         case '8': OutputFunction.append ("1000"); break;
         case '9': OutputFunction.append ("1001"); break;
         case 'a': OutputFunction.append ("1010"); break;
         case 'A': OutputFunction.append ("1010"); break;
         case 'b': OutputFunction.append ("1011"); break;
         case 'B': OutputFunction.append ("1011"); break;
         case 'c': OutputFunction.append ("1100"); break;
         case 'C': OutputFunction.append ("1100"); break;
         case 'd': OutputFunction.append ("1101"); break;
         case 'D': OutputFunction.append ("1101"); break;
         case 'e': OutputFunction.append ("1110"); break;
         case 'E': OutputFunction.append ("1110"); break;
         case 'f': OutputFunction.append ("1111"); break;
         case 'F': OutputFunction.append ("1111"); break;
      }
   }
   return OutputFunction;
}

string XOR(string Str1, string Str2)
{
   OutputS="";
   for(int i=0;i<Str1.length();i++)
   {if(Str1.at(i)!=Str2.at(i)){OutputS += "1";}
   else{OutputS += "0";}}return(OutputS);
}

void Divide_String_To_Block (string Str, int Int)
{
   for(int i=0;i<Str.length()/Int;i++){OutputArrayS[i]="";
   for(int j=0;j<Int;j++){OutputArrayS[i]+=Str.at(i*Int+j);}}
}

void Divide_String_To_Block_Reverse (string Str, int Int)
{
   for(int i=0;i<Str.length()/Int;i++){OutputArrayS[i]="";
   for(int j=0;j<Int;j++){OutputArrayS[i]+=Str.at(i*Int+(Int-j-1));}}
}

```

```
86 int Binary_Decimal(string Str, int Int)
87 {
88    OutputI=0;for(int i=0;i<Int;i++)
89    {OutputI=OutputI+(Str.at(i)-48)*(int)pow(2.0,Int-1-i);}
90    return(OutputI);
91 }
92
93 string Decimal_To_Binary(unsigned int Int1, int Int2)
94 {
95    kira=0;OutputS="";
96 ulang:
97    kira++;ArrayS[kira]=Int1%2+48;
98    if(Int1/2!=0){Int1=Int1/2;goto ulang;}
99    for(int i=0;i<Int2-kira;i++){OutputS=OutputS+'0';}
100   for(int i=kira;i>0;i--)
101   {OutputS=OutputS+ArrayS[i];}return(OutputS);
102 }
103
104 string Shift_Left(string Str, int Int)
105 {
106   OutputS="";for(int i=0;i<Str.length();i++)
107   {OutputS+=Str.at((Str.length()+Int+i)%Str.length());}
108   return(OutputS);
109 }
110
111 ////////////////////////////////////////////////////////////
112 //                    ALGORITHM CODE                      //
113 //============================================================//
114 //                     KEY SCHEDULE                       //
115 //============================================================//
116 void Row_Key(string masterkey)
117 {
118   Divide_String_To_Block_Reverse (masterkey, 32);
119   for(int i=0;i<4;i++){RowKey[i]=OutputArrayS[3-i];}
120 }
121
122 void Sub_Key(int Int)
123 {
124   SubKey[Int]="";for(int j=0;j<4;j++)
125   {for(int i=16;i<32;i++){tempStr=RowKey[3-j].at(i);
126   SubKey[Int]+=tempStr;}}
127 }
128
129 void Key_Sub_Column()
130 {
131   for(int k=0;k<8;k++)
132   {
133     tempStr="";for(int j=0;j<4;j++)
134     {tempStr+=RowKey[3-j].at(32-8+k);}SubColumn_In[k]=tempStr;
135     Binary_Decimal(SubColumn_In[k],4);tempInt=S_Box[OutputI];
136     SubColumn_Out[k]=Decimal_To_Binary(tempInt, 4);
```

```
137      for(int j=0;j<4;j++)
138      {
139         tempStr+=RowKey[j].at(32-8+k);
140         RowKey[3-j].at(32-8+k)=SubColumn_Out[k].at(j);
141      }
142   }
143 }
144
145 void Row_transformation()
146 {
147   Shift_Left(RowKey[0], 8);RowKeyNew[0]=OutputS;
148   XOR(RowKeyNew[0], RowKey[1]);RowKeyNew[0]=OutputS;
149   RowKeyNew[1]=RowKey[2];
150   Shift_Left(RowKey[2], 16);RowKeyNew[2]=OutputS;
151   XOR(RowKeyNew[2], RowKey[3]);RowKeyNew[2]=OutputS;
152   RowKeyNew[3]=RowKey[0];
153   for(int i=0;i<4;i++){RowKey[i]=RowKeyNew[i];}
154 }
155
156 void keyschedule(string masterkey)
157 {
158   Row_Key(masterkey);
159
160   for(int Round=0;Round<26;Round++)
161   {
162      Sub_Key(Round); if(Round==25){break;}
163      Key_Sub_Column();
164      Row_transformation();
165   }
166 }
167 //=============================================================//
168 //                          ENCRYPTION                         //
169 //=============================================================//
170 string AddRoundkey(string plaintext, int Int)
171 {
172   XOR(plaintext, SubKey[Int]);return(OutputS);
173 }
174
175 void Sub_Column()
176 {
177   Divide_String_To_Block (OutputS, 16);for(int i=0; i<4; i++)
178   {Row[3-i]=OutputArrayS[i];}
179   for(int k=0;k<16;k++)
180   {
181      tempStr="";for(int j=0;j<4;j++){tempStr+=Row[3-j].at(k);}
182      SubColumn_In[k]=tempStr;
183      Binary_Decimal(SubColumn_In[k],4);tempInt=S_Box[OutputI];
184      SubColumn_Out[k]=Decimal_To_Binary(tempInt, 4);
185
186      for(int j=0;j<4;j++)
187      {
188         tempStr+=Row[j].at(k);
189         Row[3-j].at(k)=SubColumn_Out[k].at(j);
190      }
191   }
```

```
192   OutputS="";for(int i=0;i<4;i++){OutputS+=Row[3-i];}
193 }
194
195 void DoubleRotation3D(int Int)
196 {
197   tempStr="";
198   for(int i=0;i<64;i++){tempStr+=OutputS.at(Rot_X_axis[i]);}
199   AddRoundkey(tempStr, Int);
200   tempStr="";
201   for(int i=0;i<64;i++){tempStr+=OutputS.at(Rot_Z_axis[i]);}
202   OutputS=tempStr;
203 }
204
205 // LAO-3D Design //
206 string encryption(string plaintext)
207 {
208   ROUNDS=25;
209   AddRoundkey(plaintext, 0);
210   for(int Round=1;Round<ROUNDS+1;Round++)
211   {
212      Sub_Column();
213      DoubleRotation3D(Round);
214      Ciphertext[Round]=OutputS;
215   }
216   ciphertext=OutputS;
217   ciphertext=Ciphertext[ROUNDS];
218   return(ciphertext);
219 }
220
221 string Algo(string Str1, string Str2)
222 {
223   keyschedule(Str1);encryption(Str2);
224   OutputCipher=ciphertext;
225   return(OutputCipher);
226 }
227
228 ////////////////////////////////////////////////////////////////
229 //          DATA CATEGORIES SAMPLE GENERATOR CODE               //
230 ////////////////////////////////////////////////////////////////
231 void StrictKeyAvalanche(string KeyName, int AlgoPlaintextSize,
    int AlgoKeySize,int KeyBlockSize, int TotalSample)
232 {
233   sampletext="";for(int i=0; i<AlgoPlaintextSize; i++)
234   {semen[i] = '0';sampletext=sampletext+semen[i];}
235   ifstream inFile (KeyName.c_str());getline (inFile, line);
236   ofstream write;write.open("SKA_Output.txt");
237
238   for(int j=0; j<TotalSample; j++)
239   {
240      cout<<"(1)SKA: Sample "<<j+1<<endl;
241      samplekey = line.substr(j*AlgoKeySize*KeyBlockSize,
      AlgoKeySize*KeyBlockSize);
242
```

```
243      for(int k=0; k<KeyBlockSize; k++)
244      {
245         inputkey=samplekey.substr(k*AlgoKeySize, AlgoKeySize);
246         flippedkey[0]=inputkey;
247         keyschedule(flippedkey[0]);
248
249         for(int i=1; i<AlgoKeySize+1; i++)
250         {
251            flippedkey[i]=flippedkey[0];
252            if(flippedkey[i].at(i-1)=='0')flippedkey[i].at(i-1)='1';
253            else if (flippedkey[i].at(i-1)=='1')
               flippedkey[i].at(i-1)='0';
254         }
255
256         for(int iman=0; iman<AlgoKeySize+1; iman++)
257         {
258            Algo(flippedkey[iman],sampletext);
259            CIPHERtext[iman]=OutputCipher;
260         }
261
262         for(int ii=1; ii<(AlgoKeySize+1); ii++)
263         {
264            cipher1=CIPHERtext[0];cipher2=CIPHERtext[ii];
265            FinalCipher=XOR(cipher1,cipher2);write<<FinalCipher;
266         }
267      }
268   }
269   write.close();inFile.close();
270 }
271
272 void StrictPlaintextAvalanche(string TextName, int
    AlgoPlaintextSize, int AlgoKeySize, int PlaintextBlockSize,
    int TotalSample)
273 {
274   inputkey="";for(int i=0; i<AlgoKeySize; i++)
275   {semen[i] = '0';inputkey=inputkey+semen[i];}
276   ifstream inFile (TextName.c_str());getline (inFile, line);
277   ofstream write;write.open("SPA_Output.txt");
278
279   for(int j=0; j<TotalSample; j++)
280   {
281      cout<<"(2)SPA: Sample "<<j+1<<endl;
282      sampletext = Function_HexToBin(line.substr
         (j*AlgoPlaintextSize*PlaintextBlockSize/4, AlgoPlaintextSize*
         PlaintextBlockSize/4));
283
284      for(int k=0; k<PlaintextBlockSize; k++)
285      {
286         inputtext=sampletext.substr
           (k*AlgoPlaintextSize, AlgoPlaintextSize);
287         flippedtext[0]=inputtext;
288
```

```
289         for(int i=1; i<AlgoPlaintextSize+1; i++)
290         {
291            flippedtext[i]=flippedtext[0];
292            if(flippedtext[i].at(i-1)=='0')
293            flippedtext[i].at(i-1)='1';
294            else if (flippedtext[i].at(i-1)=='1')
295            flippedtext[i].at(i-1)='0';
296         }
297
298         for(int iman=0; iman<AlgoPlaintextSize+1; iman++)
299         {Algo(inputkey,flippedtext[iman]);
300         CIPHERtext[iman]=OutputCipher;}
301
302         for(int i=1; i<AlgoPlaintextSize+1; i++)
303         {
304            cipher1=CIPHERtext[0];cipher2=CIPHERtext[i];
305            FinalCipher=XOR(cipher1, cipher2);
306            write<<FinalCipher;
307         }
308      }
309   }
310   write.close();inFile.close();
311 }
312
313 void PlaintextCiphertextCorrelation(string TextName, string KeyName,
    int AlgoPlaintextSize, int AlgoKeySize, int PlaintextBlockSize,
    int TotalSample)
314 {
315   ifstream inFile (KeyName.c_str());getline (inFile, line);
316   ifstream inFile2 (TextName.c_str());getline (inFile2, line2);
317   ofstream write;write.open("PCC_Output.txt");
318
319   for(int j=0; j<TotalSample; j++)
320   {
321      cout<<"(3)PCC: Sample "<<j+1<<endl;
322      sampletext=Function_HexToBin (line2.substr(j*AlgoPlaintextSize*
       PlaintextBlockSize/4,AlgoPlaintextSize*PlaintextBlockSize/4));
323      samplekey=line.substr(j*AlgoKeySize,AlgoKeySize);
324
325      for(int k=0; k<PlaintextBlockSize; k++)
326      {
327         inputkey=samplekey; cipher1=sampletext.substr
          (k*AlgoPlaintextSize, AlgoPlaintextSize);
328         cipher2=Algo(inputkey,cipher1);
329         FinalCipher=XOR(cipher1, cipher2);write<<FinalCipher;
330      }
331   }
332   write.close();inFile.close();
333 }
334
```

```
335 void CipherBlockChaining(string KeyName,int AlgoPlaintextSize,
    int AlgoKeySize, int KeyBlockSize, int TotalSample)
336 {
337   sampletext="";for(int i=0; i<AlgoPlaintextSize; i++)
338   {semen[i]='0';sampletext=sampletext+semen[i];}iv="";
339   for(int i=0; i<AlgoPlaintextSize; i++)
340   {semen[i]='0';iv=iv+semen[i];}
341   ifstream inFile (KeyName.c_str());getline (inFile, line);
342   ofstream write;write.open("CBC_Output.txt");
343
344   for(int j=0; j<TotalSample; j++)
345   {
346      cout<<"(4)CBC: Sample "<<j+1<<endl;
347      inputkey = line.substr(j*AlgoKeySize, AlgoKeySize);
348
249      for(int k=0; k<KeyBlockSize; k++)
350      {
351         plainIV=XOR(sampletext,iv);
352         FinalCipher=Algo(inputkey,plainIV);
353         write<<FinalCipher;iv=FinalCipher;
354      }
355   }write.close();inFile.close();
356 }
357
358 void RandomPlaintextRandomKey(string TextName, string KeyName,
    int AlgoPlaintextSize, int AlgoKeySize, int PlaintextBlockSize,
    int TotalSample)
359 {
360   ifstream inFile (KeyName.c_str());getline (inFile, line);
361   ifstream inFile2 (TextName.c_str());getline (inFile2, line2);
362   ofstream write;write.open("RPRK_Output.txt");
363
364   for(int j=0; j<TotalSample; j++)
365   {
366      cout<<"(5)RPRK: Sample "<<j+1<<endl;
367      samplekey=line.substr(j*AlgoKeySize, AlgoKeySize);
368      sampletext=Function_HexToBin(line2.substr(j*AlgoPlaintextSize*
         PlaintextBlockSize/4, AlgoPlaintextSize*PlaintextBlockSize/4));
369
370      for(int k=0; k<PlaintextBlockSize; k++)
371      {
372         inputkey=samplekey;inputtext=sampletext.substr
           (k*AlgoPlaintextSize, AlgoPlaintextSize);
373         FinalCipher=Algo(inputkey,inputtext); write<<FinalCipher;
374      }
375   }
376   write.close();inFile.close();
377 }
378
379 void LowDensityKeys(string TextName, int AlgoPlaintextSize,
    int AlgoKeySize, int PlaintextBlockSize, int TotalSample)
380 {
381   inputkey="";for(int i=0; i<AlgoKeySize; i++) {semen[i]='0';
382   inputkey=inputkey+semen[i];}
383
```

```
384   for(int i=0; i<AlgoKeySize; i++)
385   {
386      key1bit[i]=inputkey;
387      if(key1bit[i].at(i)=='0')key1bit[i].at(i)='1';
388      else if(key1bit[i].at(i)=='1')key1bit[i].at(i)='0';
389   }
390   asal=inputkey;para=0;
391
392   for(int a=0; a<AlgoKeySize-1; a++)
393   {
394      if(asal.at(a)=='0') asal.at(a)='1';
395      for(int b=a+1; b<AlgoKeySize; b++)
396      {
397         if(asal.at(b)=='0') asal.at(b)='1';key2bit[para]=asal;
398         para++;asal.at(b)='0';}asal.at(a)='0';
399      }
400
401   InputKunci[0]=inputkey;
402   for(int i=1;i<AlgoKeySize+1;i++){InputKunci[i]=key1bit[i-1];}
403   for(int i=AlgoKeySize+1;i<PlaintextBlockSize;i++)
404   {InputKunci[i]=key2bit[i-(AlgoKeySize+1)];}
405   ifstream inFile (TextName.c_str());getline (inFile, line);
406   ofstream write;write.open("LDK_Output.txt");
407
408   for(int j=0; j<TotalSample; j++)
409   {
410      cout<<"(6)LDK: Sample "<<j+1<<endl;
411      sampletext=Function_HexToBin
412      (line.substr(j*AlgoPlaintextSize/4, AlgoPlaintextSize/4));
413
414      for(int k=0; k<PlaintextBlockSize; k++)
415      {
416         inputtext=sampletext;
417         FinalCipher=Algo(InputKunci[k],inputtext);
418         write<<FinalCipher;
419      }
420   }write.close();inFile.close();
421 }
422
423 void HighDensityKeys(string TextName, int AlgoPlaintextSize,
424 int AlgoKeySize, int PlaintextBlockSize, int TotalSample)
425 {
426   inputkey="";for(int i=0; i<AlgoKeySize; i++)
427   {semen[i] = '1';inputkey=inputkey+semen[i];}
428
429   for(int i=0; i<AlgoKeySize; i++)
430   {
431      key1bit[i]=inputkey;
432      if(key1bit[i].at(i)=='1')key1bit[i].at(i)='0';
433      else if (key1bit[i].at(i)=='0')key1bit[i].at(i)='1';
434   }
435   asal=inputkey;para=0;
436
```

```
437   for(int a=0; a<AlgoKeySize-1; a++)
438   {
439      if(asal.at(a)=='1')asal.at(a)='0';
440      for(int b=a+1;b<AlgoKeySize; b++)
441      {
442         if(asal.at(b)=='1') asal.at(b)='0';key2bit[para]=asal;
443         para++;asal.at(b)='1';
444      }
445      asal.at(a)='1';
446   }
447
448   InputKunci[0]=inputkey;for(int i=1; i<AlgoKeySize+1; i++)
449   {InputKunci[i]=key1bit[i-1];}
450   for(int i=AlgoKeySize+1; i<PlaintextBlockSize; i++)
451   {InputKunci[i]=key2bit[i-(AlgoKeySize+1)];}
452   ifstream inFile (TextName.c_str());getline (inFile, line);
453   ofstream write;write.open("HDK_Output.txt");
454
455   for(int j=0; j<TotalSample; j++)
456   {
457      cout<<"(8)HDK: Sample "<<j+1<<endl;
458      sampletext=Function_HexToBin
       (line.substr(j*AlgoPlaintextSize/4, AlgoPlaintextSize/4));
459
460      for(int k=0; k<PlaintextBlockSize; k++)
461      {
462         inputtext=sampletext;
463         FinalCipher=Algo(InputKunci[k],inputtext);
464         write<<FinalCipher;
465      }
466   }write.close();inFile.close();
467 }
468
469 void LowDensityPlaintext(string KeyName, int AlgoPlaintextSize,
    int AlgoKeySize, int KeyBlockSize, int TotalSample)
470 {
471   inputtext="";for(int i=0; i<AlgoPlaintextSize; i++)
472   {semen[i]='0';inputtext=inputtext+semen[i];}
473
474   for(int i=0;i<AlgoPlaintextSize;i++)
475   {
476      text1bit[i]=inputtext;
477      if(text1bit[i].at(i)=='0')text1bit[i].at(i)='1';
478      else if(text1bit[i].at(i)=='1')text1bit[i].at(i)='0';
479   }
480   asal=inputtext;para=0;
481
482   for(int a=0; a<AlgoPlaintextSize-1; a++)
483   {
484      if(asal.at(a)=='0') asal.at(a)='1';
485      for(int b=a+1; b<AlgoPlaintextSize; b++)
486      {
487         if(asal.at(b)=='0') asal.at(b)='1';main2bit[para]=asal;
488         para++;asal.at(b)='0';
489      }
```

```
490      asal.at(a)='0';
491   }
492
493   InputTeks[0]=inputtext;for(int i=1;i<AlgoPlaintextSize+1;i++)
494   {InputTeks[i]=text1bit[i-1];}
495   for(int i=AlgoPlaintextSize+1;i<KeyBlockSize;i++)
496   {InputTeks[i]=main2bit[i-(AlgoPlaintextSize+1)];}
497   ifstream inFile (KeyName.c_str());getline (inFile,line);
498   ofstream write;write.open("LDP_Output.txt");
499
500   for(int j=0; j<TotalSample; j++)
501   {
502      cout<<"(7)LDP: Sample "<<j+1<<endl;
503      samplekey=line.substr(j*AlgoKeySize, AlgoKeySize);
504
505      for(int k=0; k<KeyBlockSize; k++)
506      {
507         inputkey=samplekey;FinalCipher=Algo(inputkey,InputTeks[k]);
508         write<<FinalCipher;
509      }
510   }write.close();inFile.close();
511 }
512
513 void HighDensityPlaintext(string KeyName,int AlgoPlaintextSize,
514 int AlgoKeySize, int KeyBlockSize, int TotalSample)
515 {
516   inputtext="";for(int i=0; i<AlgoPlaintextSize; i++)
517   {semen[i]='1';inputtext=inputtext+semen[i];}
518
519   for(int i=0;i<AlgoPlaintextSize;i++)
520   {
521      text1bit[i]=inputtext;
522      if(text1bit[i].at(i)=='1')text1bit[i].at(i)='0';
523      else if(text1bit[i].at(i)=='0')text1bit[i].at(i)='1';
524   }
525   asal=inputtext;para=0;
526
527   for(int a=0;a<AlgoPlaintextSize-1;a++)
528   {
529      if(asal.at(a)=='1')asal.at(a)='0';
530      for(int b=a+1;b<AlgoPlaintextSize; b++)
531      {
532         if(asal.at(b)=='1') asal.at(b)='0';
533         main2bit[para]=asal;para++;asal.at(b)='1';
534      }
535      asal.at(a)='1';
536   }
537
538   InputTeks[0]=inputtext;for(int i=1;i<AlgoPlaintextSize+1;i++)
539   {InputTeks[i]=text1bit[i-1];}
540   for(int i=AlgoPlaintextSize+1;i<KeyBlockSize;i++)
541   {InputTeks[i]=main2bit[i-(AlgoPlaintextSize+1)];}
542   ifstream inFile (KeyName.c_str());getline (inFile, line);
543   ofstream write;write.open("HDP_Output.txt");
544
```

```
545   for(int j=0; j<TotalSample; j++)
546   {
547      cout<<"(9)HDP: Sample "<<j+1<<endl;
548      samplekey=line.substr(j*AlgoKeySize, AlgoKeySize);
549
550      for(int k=0; k<KeyBlockSize; k++)
551      {
552         inputkey=samplekey;FinalCipher=Algo(inputkey,InputTeks[k]);
553         write<<FinalCipher;
554      }
555   }
556   write.close();inFile.close();
557 }
558
559 int main()
560 {
561   ////////////////////////////////////////////////////////////
562   //                    INPUTS FOR DATA CATEGORIES          //
563   ////////////////////////////////////////////////////////////
564   int TotalSample = 1000;
565   int AlgoKeySize = 128;
566   int AlgoPlaintextSize = 64;
567   string Nonce = "0100000101000010010001000101010101001100010000010
      1001100010010010100011001011010010000010100101101000001010100100
      1
      00100101000001";
568   ////////////////////////////////////////////////////////////
569   //                     DATA CATEGORY SELECTION            //
570   ////////////////////////////////////////////////////////////
571   cout<<" 1 - Strict Key Avalanche"<<endl;
572   cout<<" 2 - Strict Plaintext Avalanche"<<endl;
573   cout<<" 3 - Plaintext Ciphertext Correlation"<<endl;
574   cout<<" 4 - Cipher Block Chaining"<<endl;
575   cout<<" 5 - Random Plaintext Random Key"<<endl;
576   cout<<" 6 - Low Density Keys"<<endl;
577   cout<<" 7 - Low Density Plaintext"<<endl;
578   cout<<" 8 - High Density Keys"<<endl;
579   cout<<" 9 - High Density Plaintext"<<endl;
580   cout<<" 10 - All data categories"<<endl<<endl;
581   cout<<"Choose Data Category: ";cin>>DataCategory;
582   if(DataCategory==10){mula=1; akhir=10;}
583   if(DataCategory<10){mula=DataCategory; akhir=DataCategory+1;}
584
585   for(int i=mula; i<akhir; i++)
586   {
587      if (i==1)
588      {
589         cout<<endl<<"(1.Strict Key Avalanche)"<<endl<<endl;
590         KeyName = "SKA(Key)_inp_128_bin.txt"; KeyBlockSize = 123;
591         StrictKeyAvalanche(KeyName, AlgoPlaintextSize, AlgoKeySize,
            KeyBlockSize, TotalSample);
592      }
593
```

```
594      if (i==2)
595      {
596         cout<<endl<<"(2.Strict Plaintext Avalanche)"<<endl<<endl;
597         TextName = "SPA(Plaintext)_inp_128_hex.txt";
598         PlaintextBlockSize = 245;
599         StrictPlaintextAvalanche(TextName, AlgoPlaintextSize,
            AlgoKeySize, PlaintextBlockSize, TotalSample);
600      }
601
602      if (i==3)
603      {
604         cout<<endl<<"(3.Plaintext Ciphertext
            Correlation)"<<endl<<endl;
605         KeyName = "PCC(Key)_inp_128_bin.txt";
606         TextName = "PCC(Plaintext)_inp_128_hex.txt";
607         PlaintextBlockSize = 15625;
608         PlaintextCiphertextCorrelation(TextName, KeyName,
            AlgoPlaintextSize, AlgoKeySize, PlaintextBlockSize,
            TotalSample);
609      }
610
611      if (i==4)
612      {
613         cout<<endl<<"(4.Cipher Block Chaining)"<<endl<<endl;
614         KeyName = "CBC(Key)_inp_128_bin.txt"; KeyBlockSize = 15625;
615         CipherBlockChaining(KeyName, AlgoPlaintextSize, AlgoKeySize,
            KeyBlockSize, TotalSample);
616      }
617
618      if (i==5)
619      {
620         cout<<endl<<"(5.Random Plaintext Random Key)"<<endl<<endl;
621         KeyName = "RPRK(Key)_inp_128_bin.txt";
622         TextName = "RPRK(Plaintext)_inp_128_hex.txt";
623         PlaintextBlockSize = 15625;
624         RandomPlaintextRandomKey(TextName, KeyName,
            AlgoPlaintextSize, AlgoKeySize, PlaintextBlockSize,
            TotalSample);
625      }
626
627      if (i==6)
628      {
629         cout<<endl<<"(6.Low Density Keys)"<<endl<<endl;
630         KeyName = "LDK(Plaintext)_inp_128_hex.txt";
631         PlaintextBlockSize=8257;
632         LowDensityKeys(KeyName, AlgoPlaintextSize, AlgoKeySize,
            PlaintextBlockSize, TotalSample);
633      }
634
635      if (i==7)
636      {
637         cout<<endl<<"(7.Low Density Plaintext)"<<endl<<endl;
638         KeyName = "LDP(Key)_inp_128_bin.txt"; KeyBlockSize=2081;
639         LowDensityPlaintext(KeyName, AlgoPlaintextSize, AlgoKeySize,
            KeyBlockSize, TotalSample);
640      }
```

```
641
642      if (i==8)
643      {
644         cout<<endl<<"(8.High Density Keys)"<<endl<<endl;
645         TextName = "HDK(Plaintext)_inp_128_hex.txt";
646         PlaintextBlockSize=8257;
647         HighDensityKeys(TextName, AlgoPlaintextSize,
            AlgoKeySize, PlaintextBlockSize, TotalSample);
648      }
649
650      if (i==9)
651      {
652         cout<<endl<<"(9.High Density Plaintext)"<<endl<<endl;
653         KeyName = "HDP(Key)_inp_128_bin.txt"; KeyBlockSize=2081;
654         HighDensityPlaintext(KeyName,AlgoPlaintextSize, AlgoKeySize,
            KeyBlockSize, TotalSample);
655      }
656   }
657
658   return 0;
659 }
```

APPENDIX G

Source code of Differential Cryptanalysis for LAO-3D Lightweight Block Cipher

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <math.h>
using namespace std;

 int OutDec,Active,KiraBit,OutputI,berkali,KiraRound,DA,DB,
   JumSemuaActive;
int TagActSbox[20],TagInDDT[20],TagJumActSb[20],KiraSelected[99],
   InRotZ[20],InRotX[20],OutRotZ[20],kiraR[999],KirPos[20];
int KirHex[20][20],TagJumOut[20][20],TagOutDDT[20][20],Jay[20][20],
   TagValDDT[20][20][20];
string stA,stB,stC,OutBin,InpString,Out1,temp,tempStr;
string TagHexDarabCarry[16],TagHexDarab[20][9999],TagHex[20][20][20];
char OutHex,cont;
float DarabCarry[16],Darab[20][9999],TagPrDDT[20][20][20];

// BOX //

int S_Box[20] = {12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2};

int Rot_X_axis[64]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,28,24,20,
   16,29,25,21,17,30,26,22,18,31,27,23,19,47,46,45,44,43,42,41,40,39,
   38,37,36,35,34,33,32,51,55,59,63,50,54,58,62,49,53,57,61,48,52,56,
   60};

int Rot_Z_axis[64]={0,13,62,51,4,29,58,35,8,45,54,19,12,61,50,3,16,
   9,46,55,20,25,42,39,24,41,38,23,28,57,34,7,32,5,30,59,36,21,26,43,40,
   37,22,27,44,53,18,11,48,1,14,63,52,17,10,47,56,33,6,31,60,49,2,15};

int DDT[20][20] = {{16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,4,0,0,0,4,0,4,0,0,0,4,0,0},{0,0,0,2,0,4,2,0,0,0,2,0,2,2,2,0},
   {0,2,0,2,2,0,4,2,0,0,2,2,0,0,0,0},{0,0,0,0,0,4,2,2,0,2,2,0,2,0,2,0},
   {0,2,0,0,2,0,0,0,0,2,2,2,4,2,0,0},{0,0,2,0,0,0,2,0,2,0,0,4,2,0,0,4},
   {0,4,2,0,0,0,2,0,2,0,0,0,2,0,0,4},{0,0,0,2,0,0,0,2,0,2,0,4,0,2,0,4},
   {0,0,2,0,4,0,2,0,2,0,0,0,2,0,4,0},{0,0,2,2,0,4,0,0,2,0,2,0,0,2,2,0},
   {0,2,0,0,2,0,0,0,4,2,2,2,0,2,0,0},{0,0,2,0,0,4,0,2,2,2,2,0,0,0,2,0},
   {0,2,4,2,2,0,0,2,0,0,2,2,0,0,0,0},{0,0,2,2,0,0,2,2,2,2,0,0,2,2,0,0},
   {0,4,0,0,4,0,0,0,0,0,0,0,0,0,4,4}};
```

```
27 string Dec2Bin(int A)
28 {
29    if(A==0){OutBin="0000";} if(A==1){OutBin="0001";}
30    if(A==2){OutBin="0010";} if(A==3){OutBin="0011";}
31    if(A==4){OutBin="0100";} if(A==5){OutBin="0101";}
32    if(A==6){OutBin="0110";} if(A==7){OutBin="0111";}
33    if(A==8){OutBin="1000";} if(A==9){OutBin="1001";}
34    if(A==10){OutBin="1010";}if(A==11){OutBin="1011";}
35    if(A==12){OutBin="1100";}if(A==13){OutBin="1101";}
36    if(A==14){OutBin="1110";}if(A==15){OutBin="1111";}return(OutBin);
37 }
38
39 char Dec2Hex(int A)
40 {
41    if(A==0){OutHex='0';} if(A==1){OutHex='1';} if(A==2){OutHex='2';}
42    if(A==3){OutHex='3';} if(A==4){OutHex='4';} if(A==5){OutHex='5';}
43    if(A==6){OutHex='6';} if(A==7){OutHex='7';} if(A==8){OutHex='8';}
44    if(A==9){OutHex='9';} if(A==10){OutHex='A';}if(A==11){OutHex='B';}
45    if(A==12){OutHex='C';}if(A==13){OutHex='D';}if(A==14){OutHex='E';}
46    if(A==15){OutHex='F';}return(OutHex);
47 }
48
49 int Char2Dec(char B)
50 {
51    if(B=='0'){OutDec=0;} if(B=='1'){OutDec=1;} if(B=='2'){OutDec=2;}
52    if(B=='3'){OutDec=3;} if(B=='4'){OutDec=4;} if(B=='5'){OutDec=5;}
53    if(B=='6'){OutDec=6;} if(B=='7'){OutDec=7;} if(B=='8'){OutDec=8;}
54    if(B=='9'){OutDec=9;} if(B=='A'){OutDec=10;}if(B=='B'){OutDec=11;}
55    if(B=='C'){OutDec=12;}if(B=='D'){OutDec=13;}if(B=='E'){OutDec=14;}
56    if(B=='F'){OutDec=15;}return(OutDec);
57 }
58
59 int Bin2Dec(string Str, int Int)
60 {
61    OutputI=0;for(int i=0;i<Int;i++)
62    {OutputI=OutputI+(Str.at(i)-48)*(int)pow(2.0,Int-1-i);}
63    return(OutputI);
64 }
65
66 char XOR_Hex (char A, char B)
67 {
68    stA=Dec2Bin(Char2Dec(A)); stB=Dec2Bin(Char2Dec(B)); stC="";
69    for(int i=0;i<4;i++)
70    {
71       if(stA.at(i)==stB.at(i)){temp='0';}
72       if(stA.at(i)!=stB.at(i)){temp='1';}stC=stC+temp;
73    }
74    OutHex=Dec2Hex(Bin2Dec(stC, 4));return(OutHex);
75 }
76
```

```
77 int main()
78 {
79    cout<<"Differential Cryptanalysis"<<endl<<endl;
80    InpString="0000000000000001";
81
82 SINI:
83    cout<<"Input: "<<InpString<<endl;
84    Active=0;
85
86    for(int i=0;i<16;i++)
87    {
88       if((int)InpString.at(i) != '0')
89       {
90          Char2Dec(InpString.at(i));Dec2Bin(OutDec);Active++;
91          TagActSbox[Active]=i+1;TagInDDT[TagActSbox[Active]]=OutDec;
92          cout<<"Active S-box("<<TagActSbox[Active]<<") "<<
            TagInDDT[TagActSbox[Active]]<<" = "<<OutBin<<endl;
93       }
94    }
95    cout<<endl<<"Active S-box: "<<Active<<endl<<endl;
96    JumSemuaActive=JumSemuaActive+Active;
97
98    for(int i=1;i<Active+1;i++)
99    {
100      cout<<"================"<<endl;cout<<"Output S-box("<<
101      TagActSbox[i]<<")"<<endl;cout<<"================"<<endl;
102      TagJumActSb[i]=0;
103
104      for(int j=0;j<16;j++)
105      {
106         TagValDDT[TagActSbox[i]][TagInDDT[TagActSbox[i]]][j]=
            DDT[TagInDDT[TagActSbox[i]]][j];
107
108         if(TagValDDT[TagActSbox[i]][TagInDDT[TagActSbox[i]]][j]!=0)
109         {
110            TagJumOut[i][j]=j;TagJumActSb[i]++;
111            TagOutDDT[i][TagJumActSb[i]]=j;KirPos[i]++;
112            Jay[i][KirPos[i]]=j;Dec2Bin(j);
113            TagPrDDT[TagActSbox[i]][TagInDDT[TagActSbox[i]]]
               [TagOutDDT[i][TagJumActSb[i]]]=
               (float)TagValDDT[TagActSbox[i]]
               [TagInDDT[TagActSbox[i]]][TagJumOut[i][j]]/16;
114            cout<<"TagPrDDT["<<TagActSbox[i]<<"]["<<
               TagInDDT[TagActSbox[i]]<<"]["<<
               TagOutDDT[i][TagJumActSb[i]];
115            cout<<"]: "<<TagPrDDT[TagActSbox[i]]
               [TagInDDT[TagActSbox[i]]][TagOutDDT[i][TagJumActSb[i]]];
116            cout<<" ("<<TagValDDT[TagActSbox[i]]
               [TagInDDT[TagActSbox[i]]][TagOutDDT[i][TagJumActSb[i]]]<<
               "/16) ---> "<<OutBin<<endl;
117
```

```
118            KiraBit=0;
119            for(int k=0;k<4;k++)
120            {
121               if(OutBin.at(k)=='1')
122               {
123                  KiraBit++;InRotX[KiraBit]=(TagActSbox[i]-1)*4+k;
124                  InRotZ[KiraBit]=Rot_X_axis[InRotX[KiraBit]];
125                  OutRotZ[KiraBit]=Rot_Z_axis[InRotZ[KiraBit]];
126               }
127            }
128
129            Out1="00000000000000000000000000000000000000000000000000000000
               0000000000000";
130            for(int m=1;m<KiraBit+1;m++){Out1.at(OutRotZ[m])='1';}
131            TagHex[TagActSbox[i]][TagInDDT[TagActSbox[i]]]
               [TagOutDDT[i][TagJumActSb[i]]]="";
132
133            for(int n=0;n<16;n++)
134            {
135               temp="";
136               for(int p=0;p<4;p++){temp=temp+Out1.at(n*4+p);}
137               Dec2Hex(Bin2Dec(temp, 4));
138               TagHex[TagActSbox[i]][TagInDDT[TagActSbox[i]]]
                  [TagOutDDT[i][TagJumActSb[i]]]=TagHex[TagActSbox[i]]
                  [TagInDDT[TagActSbox[i]]][TagOutDDT[i]
                  [TagJumActSb[i]]]+OutHex;
139            }
140
141            cout<<"TagHex["<<TagActSbox[i]<<"]["<<
               TagInDDT[TagActSbox[i]]<<"]["<<
               TagOutDDT[i][TagJumActSb[i]]<<"]; "<<
               TagHex[TagActSbox[i]][TagInDDT[TagActSbox[i]]]
               [TagOutDDT[i][TagJumActSb[i]]]<<endl<<endl;
142         }
143      }
144   }
145
146   for(int i=1;i<Active+1;i++)
147   {
148      kiraR[i]=0;KiraSelected[i]=0;
149      cout<<"================"<<endl;cout<<i<<") S-box("<<
         TagActSbox[i]<<")"<<endl;cout<<"================"<<endl;
150      if(i==1){berkali=1;}
151      if(i>1){berkali=KiraSelected[i-1];}
152
153      for(int j=1;j<TagJumActSb[i]+1;j++)
154      {
155         for(int k=1;k<berkali+1;k++)
156         {
157            kiraR[i]++;
158            TagHexDarab[i][kiraR[i]]=TagHex[TagActSbox[i]]
               [TagInDDT[TagActSbox[i]]][TagOutDDT[i][j]];
159
```

```
160             if(i==1)
161             {
162                Darab[i-1][k]=1;
163                Darab[i][kiraR[i]]=TagPrDDT[TagActSbox[i]]
                   [TagInDDT[TagActSbox[i]]][TagOutDDT[i][j]]*
                   Darab[i-1][k];
164                KiraSelected[i]++;KirHex[i][kiraR[i]]=0;
165
166                for(int h=0;h<16;h++)
167                {
168                   if(TagHexDarab[i][kiraR[i]].at(h)!='0')
169                   {KirHex[i][kiraR[i]]++;}
170                }
171
172                cout<<"TagPrDDT["<<TagActSbox[i]<<"]["<<
                   TagInDDT[TagActSbox[i]]<<"]["<<TagOutDDT[i][j]<<
                   "]: "<<TagPrDDT[TagActSbox[i]]
                   [TagInDDT[TagActSbox[i]]][TagOutDDT[i][j]];
173                cout<<" X Darab["<<i-1<<"]["<<k<<"]: "<<
                   Darab[i-1][k]<<" = Darab["<<i<<"]["<<kiraR[i]<<"]: "<<
                   Darab[i][kiraR[i]];
174                cout<<" -> TagHexDarab["<<i<<"]["<<kiraR[i]<<
                   "]: "<<TagHexDarab[i][kiraR[i]]<<" (ActiveSbox:"<<
                   KirHex[i][kiraR[i]]<<")"<<endl;
175
176                if(j==TagJumActSb[i] && i!=Active)
177                {
178                   cout<<endl<<"Darab[A][B]"<<endl;
179                   cout<<"Enter [A]: "; cin>>DA;cout<<"Enter [B]: ";
180                   cin>>DB;
181                   TagHexDarab[DA][kiraR[DB]]=TagHexDarab[i]
                      [kiraR[i]];
182                   KiraSelected[i]=1;
183                }
184             }
185
186             if(i>1)
187             {
188                Darab[i-1][k]=Darab[DA][DB];
189                Darab[i][kiraR[i]]=TagPrDDT[TagActSbox[i]]
                   [TagInDDT[TagActSbox[i]]][TagOutDDT[i][j]]*
                   Darab[i-1][k];
190                KiraSelected[i]++;
191                Darab[i][KiraSelected[i]]=Darab[i][kiraR[i]];
192                cout<<"TagPrDDT["<<TagActSbox[i]<<"]["<<
                   TagInDDT[TagActSbox[i]]<<"]["<<TagOutDDT[i][j]<<
                   "]: "<<TagPrDDT[TagActSbox[i]]
                   [TagInDDT[TagActSbox[i]]][TagOutDDT[i][j]];
193                cout<<" X Darab["<<DA<<"]["<<DB<<"]: "<<
                   Darab[DA][DB]<<" = Darab["<<i<<"]["<<KiraSelected[i]<<
                   "]: "<<Darab[i][KiraSelected[i]];
194                tempStr="";
195
```

```
196               for(int f=0;f<16;f++)
197               {
198                  XOR_Hex (TagHexDarab[i][kiraR[i]].at(f),
                     TagHexDarab[DA][DB].at(f)); tempStr=tempStr+OutHex;
199               }
200
201               TagHexDarab[i][KiraSelected[i]]=tempStr;
202               KirHex[i][KiraSelected[i]]=0;
203
204               for(int h=0;h<16;h++)
205               {
206                  if(TagHexDarab[i][KiraSelected[i]].at(h)!='0')
207                  {KirHex[i][KiraSelected[i]]++;}
208               }
209
210               cout<<" --> TagHexDarab["<<i<<"]["<<KiraSelected[i]<<
                  "] = "<<TagHexDarab[i][KiraSelected[i]]<<
                  " (ActiveSbox:"<<KirHex[i][KiraSelected[i]]<<
                  ")"<<endl;
211
212               if(j==TagJumActSb[i] && i!=Active)
213               {
214                  cout<<endl<<"Darab[A][B]"<<endl;
215                  cout<<"Enter [A]: "; cin>>DA;
216                  cout<<"Enter [B]: "; cin>>DB; KiraSelected[i]=1;
217               }
218            }
219         }
220      }
221      cout<<endl;
222
223      if(i==Active)
224      {
225         KiraRound++;
226         cout<<endl<<"=======> Round "<<KiraRound<<" <======="<<endl;
227         cout<<endl<<"Darab[A][B]"<<endl;cout<<"Enter [A]: ";
228         cin>>DA;cout<<"Enter [B]: "; cin>>DB;
229         cout<<endl<<"JumSemuaActive: "<<JumSemuaActive<<endl;
230         DarabCarry[KiraRound]=Darab[DA][DB];
231         TagHexDarabCarry[KiraRound]=TagHexDarab[DA][DB];
232
233         if(KiraRound>1)
234         {
235            DarabCarry[KiraRound]=DarabCarry[KiraRound-1]*
               Darab[DA][DB];
236         }
237
238         cout<<endl<<"Probability: "<<DarabCarry[KiraRound]<<
239         endl<<"Output: "<<TagHexDarabCarry[KiraRound]<<endl<<endl;
240
```

```
241         if(DarabCarry[KiraRound]<1.08420217e-19)
242         {
243            cout<<"The highest probability of difference propagation
               at round "<<KiraRound<<" is "<<DarabCarry[KiraRound]<<
               " which is lower than 1.08420217e-19 (2^-63)."<<endl;
244            cout<<"Therefore, it is impossible to construct an
               Effective differential distinguisher with more than "<<
               KiraRound-1<<" rounds for this algorithm."<<endl<<endl<<
               endl;
245            return 0;
246         }
247
248         cout<<"Continue? (y/n): "; cin>>cont;
249         cout<<endl<<endl<<"=======> Round "<<KiraRound+1<<
            " <======="<<endl<<endl;
250         if(cont=='y'){InpString=TagHexDarabCarry[KiraRound];
251         goto SINI;}
252      }
253   }
254
255   return 0;
256 }
```

APPENDIX H

Source code of Linear Cryptanalysis for LAO-3D Lightweight Block Cipher

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <math.h>
#include <cstdlib>
#include <cmath>
using namespace std;

int in,OutDec,Active,KiraBit,OutputI,berkali,KiraRound,DA,DB,
   JumSemuaActive;
int InRotZ[20],InRotX[20],OutRotZ[20],KiraSelected[99],
   TagJumActSb[20],KirPos[20],kiraR[999],TagActSbox[20],TagInLAT[20];
int TagJumOut[20][20],Jay[20][20],TagOutLAT[20][20],KirHex[20][20],
   TagValLAT[20][20][20];
string stA,stB,stC,OutBin,InpString,Out1,temp,tempStr;
string TagHex[20][20][20],TagHexDarab[20][9999],TagHexDarabCarry[16];
char OutHex,cont;
float Darab[20][9999], TagPrLAT[20][20][20],DarabCarry[16];

// BOX //

int S_Box[20] = {12,5,6,11,9,0,10,13,3,14,15,8,4,7,1,2};

int Rot_X_axis[64]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,28,24,20,
   16,29,25,21,17,30,26,22,18,31,27,23,19,47,46,45,44,43,42,41,40,39,38,
   37,36,35,34,33,32,51,55,59,63,50,54,58,62,49,53,57,61,48,52,56,60};

int Rot_Z_axis[64]={0,13,62,51,4,29,58,35,8,45,54,19,12,61,50,3,16,
   9,46,55,20,25,42,39,24,41,38,23,28,57,34,7,32,5,30,59,36,21,26,43,40,
   37,22,27,44,53,18,11,48,1,14,63,52,17,10,47,56,33,6,31,60,49,2,15};

int LAT[20][20] = {{8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
   {0,0,0,0,0,-4,0,-4,0,0,0,0,0,-4,0,4},
   {0,0,2,2,-2,-2,0,0,2,-2,0,4,0,4,-2,2},
   {0,0,2,2,2,-2,-4,0,-2,2,-4,0,0,0,-2,-2},
   {0,0,-2,2,-2,-2,0,4,-2,-2,0,-4,0,0,-2,2},
   {0,0,-2,2,-2,2,0,0,2,2,-4,0,4,0,2,2},
   {0,0,0,-4,0,0,-4,0,0,-4,0,0,4,0,0,0},
   {0,0,0,4,4,0,0,0,0,-4,0,0,0,0,4,0},
   {0,0,2,-2,0,0,-2,2,-2,2,0,0,-2,2,4,4},
   {0,4,-2,-2,0,0,2,-2,-2,-2,-4,0,-2,2,0,0},
   {0,0,4,0,2,2,2,-2,0,0,0,-4,2,2,-2,2},
   {0,-4,0,0,-2,-2,2,-2,-4,0,0,0,2,2,2,-2},
   {0,0,0,0,-2,-2,-2,-2,4,0,0,-4,-2,2,2,-2},
   {0,4,4,0,-2,-2,2,2,0,0,0,0,2,-2,2,-2},
```

```
   {0,0,2,2,-4,4,-2,-2,-2,-2,0,0,-2,-2,0,0},
   {0,4,-2,2,0,0,-2,-2,-2,2,4,0,2,2,0,0}};
28
29 string Dec2Bin(int A)
30 {
31    if(A==0){OutBin="0000";} if(A==1){OutBin="0001";}
      if(A==2){OutBin="0010";} if(A==3){OutBin="0011";}
      if(A==4){OutBin="0100";} if(A==5){OutBin="0101";}
      if(A==6){OutBin="0110";} if(A==7){OutBin="0111";}
      if(A==8){OutBin="1000";} if(A==9){OutBin="1001";}
      if(A==10){OutBin="1010";}if(A==11){OutBin="1011";}
      if(A==12){OutBin="1100";}if(A==13){OutBin="1101";}
      if(A==14){OutBin="1110";}if(A==15){OutBin="1111";}return(OutBin);
32 }
33
34 char Dec2Hex(int A)
35 {
36    if(A==0){OutHex='0';} if(A==1){OutHex='1';} if(A==2){OutHex='2';}
      if(A==3){OutHex='3';} if(A==4){OutHex='4';} if(A==5){OutHex='5';}
      if(A==6){OutHex='6';} if(A==7){OutHex='7';} if(A==8){OutHex='8';}
      if(A==9){OutHex='9';} if(A==10){OutHex='A';}if(A==11){OutHex='B';}
      if(A==12){OutHex='C';}if(A==13){OutHex='D';}if(A==14){OutHex='E';
      if(A==15){OutHex='F';}return(OutHex);
37 }
38
39 int Char2Dec(char B)
40 {
41    if(B=='0'){OutDec=0;} if(B=='1'){OutDec=1;} if(B=='2'){OutDec=2;}
      if(B=='3'){OutDec=3;} if(B=='4'){OutDec=4;} if(B=='5'){OutDec=5;}
      if(B=='6'){OutDec=6;} if(B=='7'){OutDec=7;} if(B=='8'){OutDec=8;}
      if(B=='9'){OutDec=9;} if(B=='A'){OutDec=10;}if(B=='B'){OutDec=11;}
      if(B=='C'){OutDec=12;}if(B=='D'){OutDec=13;}if(B=='E'){OutDec=14;
      if(B=='F'){OutDec=15;}return(OutDec);
42 }
43
44 int Bin2Dec(string Str, int Int)
45 {
46    OutputI=0;for(int i=0;i<Int;i++){OutputI=OutputI+(Str.at(i)-48)*
      (int)pow(2.0,Int-1-i);}return(OutputI);
47 }
48
49 char XOR_Hex (char A, char B)
50 {
51    stA=Dec2Bin(Char2Dec(A)); stB=Dec2Bin(Char2Dec(B)); stC="";
52
53    for(int i=0;i<4;i++)
54    {
55       if(stA.at(i)==stB.at(i)){temp='0';}
56       if(stA.at(i)!=stB.at(i)){temp='1';}stC=stC+temp;
57    }
58    OutHex=Dec2Hex(Bin2Dec(stC, 4));return(OutHex);
59 }
60
```

```
61 int main()
62 {
63    cout<<"Differential Cryptanalysis"<<endl<<endl;
64    InpString="0000000000000001";
65
66 SINI:
67    cout<<"Input: "<<InpString<<endl;
68    Active=0;
69
70    for(int i=0;i<16;i++)
71    {
72       if((int)InpString.at(i) != '0')
73       {
74          Char2Dec(InpString.at(i));Dec2Bin(OutDec);Active++;
75          TagActSbox[Active]=i+1;TagInLAT[TagActSbox[Active]]=OutDec;
76          cout<<"Active S-box("<<TagActSbox[Active]<<") "<<
            TagInLAT[TagActSbox[Active]]<<" = "<<OutBin<<endl;
77       }
78    }
79    cout<<endl<<"Active S-box: "<<Active<<endl<<endl;
80    JumSemuaActive=JumSemuaActive+Active;
81
82    for(int i=1;i<Active+1;i++)
83    {
84       cout<<"================"<<endl;cout<<"Output S-box("<<
          TagActSbox[i]<<")"<<endl;cout<<"================"<<endl;
85       TagJumActSb[i]=0;
86
87       for(int j=0;j<16;j++)
88       {
89          TagValLAT[TagActSbox[i]][TagInLAT[TagActSbox[i]]][j]=
           LAT[TagInLAT[TagActSbox[i]]][j];
90
91          if(TagValLAT[TagActSbox[i]][TagInLAT[TagActSbox[i]]][j]!=0)
92          {
93             TagJumOut[i][j]=j;TagJumActSb[i]++;
94             TagOutLAT[i][TagJumActSb[i]]=j;KirPos[i]++;
95             Jay[i][KirPos[i]]=j;Dec2Bin(j);
96             TagPrLAT[TagActSbox[i]][TagInLAT[TagActSbox[i]]]
               [TagOutLAT[i][TagJumActSb[i]]]=(float)TagValLAT
               [TagActSbox[i]][TagInLAT[TagActSbox[i]]]
               [TagJumOut[i][j]]/16;
97             cout<<"TagPrLAT["<<TagActSbox[i]<<"]["<<
               TagInLAT[TagActSbox[i]]<<"]["<<TagOutLAT[i]
               [TagJumActSb[i]];
98             cout<<"]: "<<TagPrLAT[TagActSbox[i]]
               [TagInLAT[TagActSbox[i]]][TagOutLAT[i][TagJumActSb[i]]];
99             cout<<" ("<<TagValLAT[TagActSbox[i]]
               [TagInLAT[TagActSbox[i]]][TagOutLAT[i][TagJumActSb[i]]]<<
               "/16) ---> "<<OutBin<<endl;
100
```

```
            KiraBit=0;
            for(int k=0;k<4;k++)
            {
               if(OutBin.at(k)=='1')
               {
                  KiraBit++;InRotX[KiraBit]=(TagActSbox[i]-1)*4+k;
                  InRotZ[KiraBit]=Rot_X_axis[InRotX[KiraBit]];
                  OutRotZ[KiraBit]=Rot_Z_axis[InRotZ[KiraBit]];
               }
            }

            Out1="00000000000000000000000000000000000000000000000000000000000000";
            for(int m=1;m<KiraBit+1;m++){Out1.at(OutRotZ[m])='1';}
            TagHex[TagActSbox[i]][TagInLAT[TagActSbox[i]]]
            [TagOutLAT[i][TagJumActSb[i]]]="";

            for(int n=0;n<16;n++)
            {
               temp="";
               for(int p=0;p<4;p++){temp=temp+Out1.at(n*4+p);}
               Dec2Hex(Bin2Dec(temp, 4));
               TagHex[TagActSbox[i]][TagInLAT[TagActSbox[i]]]
               [TagOutLAT[i][TagJumActSb[i]]]=TagHex[TagActSbox[i]]
               [TagInLAT[TagActSbox[i]]][TagOutLAT[i]
               [TagJumActSb[i]]]+OutHex;
            }

            cout<<"TagHex["<<TagActSbox[i]<<"]["<<TagInLAT
            [TagActSbox[i]]<<"]["<<TagOutLAT[i][TagJumActSb[i]]<<
            "]: "<<TagHex[TagActSbox[i]][TagInLAT[TagActSbox[i]]]
            [TagOutLAT[i][TagJumActSb[i]]]<<endl<<endl;
         }
      }
   }

   for(int i=1;i<Active+1;i++)
   {
      kiraR[i]=0;KiraSelected[i]=0;
      cout<<"================"<<endl;cout<<i<<") S-box("
      <<TagActSbox[i]<<")"<<endl;cout<<"================"<<endl;
      if(i==1){berkali=1;}if(i>1){berkali=KiraSelected[i-1];}

      for(int j=1;j<TagJumActSb[i]+1;j++)
      {
         for(int k=1;k<berkali+1;k++)
         {
            kiraR[i]++;TagHexDarab[i][kiraR[i]]=TagHex[TagActSbox[i]]
            [TagInLAT[TagActSbox[i]]][TagOutLAT[i][j]];

```

```
141             if(i==1)
142             {
143                Darab[i-1][k]=1;
144                Darab[i][kiraR[i]]=TagPrLAT[TagActSbox[i]]
                   [TagInLAT[TagActSbox[i]]][TagOutLAT[i][j]]*
                   Darab[i-1][k];
145                KiraSelected[i]++;KirHex[i][kiraR[i]]=0;
146
147                for(int h=0;h<16;h++)
                   {
148                   if(TagHexDarab[i][kiraR[i]].at(h)!='0')
149                   {KirHex[i][kiraR[i]]++;}
                   }
150
151                cout<<"TagPrLAT["<<TagActSbox[i]<<"]["<<
                   TagInLAT[TagActSbox[i]]<<"]["<<TagOutLAT[i][j]<<
                   "]: "<<TagPrLAT[TagActSbox[i]][TagInLAT
                   [TagActSbox[i]]][TagOutLAT[i][j]];
152                cout<<" X Darab["<<i-1<<"]["<<k<<"]: "<<
                   Darab[i-1][k]<<" = Darab["<<i<<"]["<<kiraR[i]<<
                   "]: "<<Darab[i][kiraR[i]];
153                cout<<" -> TagHexDarab["<<i<<"]["<<kiraR[i]<<
                   "]: "<<TagHexDarab[i][kiraR[i]]<<
                   " (ActiveSbox:"<<KirHex[i][kiraR[i]]<<")"<<endl;
154
155                if(j==TagJumActSb[i] && i!=Active)
156                {
157                   cout<<endl<<"Darab[A][B]"<<endl;cout<<
                      "Enter [A]: ";
158                   cin>>DA;cout<<"Enter [B]: ";cin>>DB;
159                   TagHexDarab[DA][kiraR[DB]]=TagHexDarab[i]
                      [kiraR[i]];
160                   KiraSelected[i]=1;
161                }
162             }
163
164             if(i>1)
165             {
166                Darab[i-1][k]=Darab[DA][DB];
167                Darab[i][kiraR[i]]=TagPrLAT[TagActSbox[i]][TagInLAT
                   [TagActSbox[i]]][TagOutLAT[i][j]]*Darab[i-1][k];
168                KiraSelected[i]++;
169                Darab[i][KiraSelected[i]]=Darab[i][kiraR[i]];
170                cout<<"TagPrLAT["<<TagActSbox[i]<<"]["<<TagInLAT
                   [TagActSbox[i]]<<"]["<<TagOutLAT[i][j]<<
                   "]: "<<TagPrLAT[TagActSbox[i]]
                   [TagInLAT[TagActSbox[i]]][TagOutLAT[i][j]];
171                cout<<" X Darab["<<DA<<"]["<<DB<<"]: "<<
                   Darab[DA][DB]<<" = Darab["<<i<<"]["<<KiraSelected[i]<<
                   "]: "<<Darab[i][KiraSelected[i]];
172
```

```
173             tempStr="";
174             for(int f=0;f<16;f++)
175             {
176                XOR_Hex (TagHexDarab[i][kiraR[i]].at(f),
                   TagHexDarab[DA][DB].at(f));
                   tempStr=tempStr+OutHex;
177             }
178
179             TagHexDarab[i][KiraSelected[i]]=tempStr;
180             KirHex[i][KiraSelected[i]]=0;
181
182             for(int h=0;h<16;h++)
183             {
184                if(TagHexDarab[i][KiraSelected[i]].at(h)!='0')
185                {KirHex[i][KiraSelected[i]]++;}}
186                cout<<" --> TagHexDarab["<<i<<"]["<<
                   KiraSelected[i]<<"] = "<<TagHexDarab[i]
                   [KiraSelected[i]]<<" (ActiveSbox:"<<
                   KirHex[i][KiraSelected[i]]<<")"<<endl;
187
188                if(j==TagJumActSb[i] && i!=Active)
189                {
190                   cout<<endl<<"Darab[A][B]"<<endl;
191                   cout<<"Enter [A]: "; cin>>DA;
192                   cout<<"Enter [B]: "; cin>>DB; KiraSelected[i]=1;
193                }
194             }
195          }
196       }
197       cout<<endl;
198
199       if(i==Active)
200       {
201          KiraRound++;
202          cout<<endl<<"=======> Round "<<KiraRound<<
             " <=======<<endl;
203          cout<<endl<<"Darab[A][B]"<<endl;cout<<"Enter [A]: ";
204          cin>>DA;cout<<"Enter [B]: "; cin>>DB;
205          DarabCarry[KiraRound]=Darab[DA][DB];
206          TagHexDarabCarry[KiraRound]=TagHexDarab[DA][DB];
207
208          if(KiraRound>1)
209          {
210             DarabCarry[KiraRound]=DarabCarry[KiraRound-1]*
               Darab[DA][DB];
211          }
212
213          cout<<"JumSemuaActive: "<<JumSemuaActive<<endl;
214          Que[i]=pow(2.0,JumSemuaActive-1);
215          Que2[i]=Que[i]*DarabCarry[KiraRound];
216          cout<<"DarabCarry["<<KiraRound<<"]: "<<
             abs(DarabCarry[KiraRound])<<endl<<"Output: "<<
             TagHexDarabCarry[KiraRound]<<endl<<endl;
217          cout<<endl<<"q:"<<Que[i]<<" x "<<DarabCarry[KiraRound]<<
             " = "<<Que2[i]<<endl;
218          cout<<"Probability: "<<0.5+Que2[i]<<endl;
```

```
219
220             if(abs(DarabCarry[KiraRound])<2.32830644e-10) //32
221             {
222                cout<<"The highest probability of linear
                   propagation at round "<<KiraRound<<" is "<<
                   DarabCarry[KiraRound]<<" which is lower than
                   2.32830644e-10 (2^-32)."<<endl;
223                cout<<"Therefore, it is impossible to construct an
                   effective linear propogation with more than "<<
                   KiraRound-1<<" rounds for this algorithm."<<endl<<
                   endl<<endl;
224                return 0;
225             }
226
227             cout<<"Continue? (y/n): "; cin>>cont;
228             cout<<endl<<endl<<"=======> Round "<<KiraRound+1<<
                " <======="<<endl<<endl;
229             if(cont=='y')
230             {InpString=TagHexDarabCarry[KiraRound];goto SINI;}
231          }
232       }
233       return 0;
234    }
235 }
```

APPENDIX I

Source code of Avalanche Effect (Correlation Coefficient, Bit Error Rate, and Key Sensitivity Tests) for LAO-3D Lightweight Block Cipher

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <cmath>
#include <cstdlib>
using namespace std;

string InputA,InputB,InputC,InStrB;
int tempIntA,tempIntB,IA[128],IB[128];
float OutDouA,OutDouB,OutDouC,OutDouD,OutDouE,InDouA,AE,Pi[128],
Ci[128],PiCi[128];

double AvalancheEffect(string InStrA, string InStrB)
{
   OutDouA=0.0; tempIntB=0;

   for(int i=0;i<InStrA.length();i++)
   {
      if(InStrA.at(i)==InStrB.at(i)){tempIntA=0;}
      if(InStrA.at(i)!=InStrB.at(i)){tempIntA=1;}
      tempIntB=tempIntB+tempIntA;
   }

   OutDouA=(float)tempIntB/InStrA.length();
   cout<<"Avalanche Effect: "<<OutDouA<<endl<<endl;
   return(OutDouA);
}

double CorrelationCoefficient (string InStrA, string InStrB,
double InDouA)
{
   OutDouA=0.0;

   for(int i=0;i<InStrA.length();i++)
   {
      if(InStrA.at(i)=='1'){IA[i]=1;}if(InStrA.at(i)=='0'){IA[i]=0;}
      if(InStrB.at(i)=='1'){IB[i]=1;}if(InStrB.at(i)=='0'){IB[i]=0;}
      Pi[i]=(double)IA[i]-InDouA; Ci[i]=(double)IB[i]-InDouA;
      PiCi[i]=Pi[i]*Ci[i]; OutDouA=OutDouA+PiCi[i];
```

```
42       OutDouB=OutDouB+Pi[i]*Pi[i]; OutDouC=OutDouC+Ci[i]*Ci[i];
43    }
44
45    OutDouD=sqrt(OutDouB)*sqrt(OutDouC); OutDouE=OutDouA/OutDouD;
46    cout<<"Correlation Coefficient: "<<OutDouE<<endl;
47    return(OutDouE);
48 }
49
50 double BitErrorRate(string InStrA, string InStrB)
51 {
52    tempIntB=0;OutDouA=0.0;
53
54    for(int i=0;i<InStrA.length();i++)
55    {
56       if(InStrA.at(i)==InStrB.at(i)){tempIntA=0;}
57       if(InStrA.at(i)!=InStrB.at(i)){tempIntA=1;}
58       tempIntB=tempIntB+tempIntA;
59    }
60
61    OutDouA=(float)tempIntB/InStrA.length();
62    cout<<"Bit Error Rate: "<<OutDouA<<" ["<<tempIntB<<" bit(s)]"<<
      endl<<endl;
63    return(OutDouA);
64 }
65
66 void KeySensitivityTest(string InStrA)
67 {
68    for(int i=0;i<InStrA.length();i++)
69    {
70       InStrB=InStrA;
71       if(InStrA.at(i)=='0'){InStrB.at(i)='1';}
72       if(InStrA.at(i)=='1'){InStrB.at(i)='0';}
73    }
74 }
75
76 int main()
77 {
78    InputA="1100000110000100011101000000100011000111110011111001111100
      010001";
79    InputB="1111100110110000101111101101100111011000110001010100111111
      001111";
80
81    cout<<"Avalanche Effect"<<endl<<endl;
82    cout<<"InputA: "<<InputA<<endl;
83    cout<<"InputB: "<<InputB<<endl<<endl;
84
85    AE=AvalancheEffect(InputA, InputB);
86    CorrelationCoefficient (InputA, InputB, AE);
87    BitErrorRate(InputA, InputB);
88    KeySensitivityTest(InputA);
89
90    return 0;
91 }
```

APPENDIX J

Samples of Data Set

The samples of data set for the cryptanalysis which includes correlation coefficient, bit error rate, key sensitivity, and randomness tests can be obtained from the following link:

https://drive.google.com/drive/folders/1fUA2Cimoos4O0Mr6CltYcZU6A1B3LE1n?usp=sharing